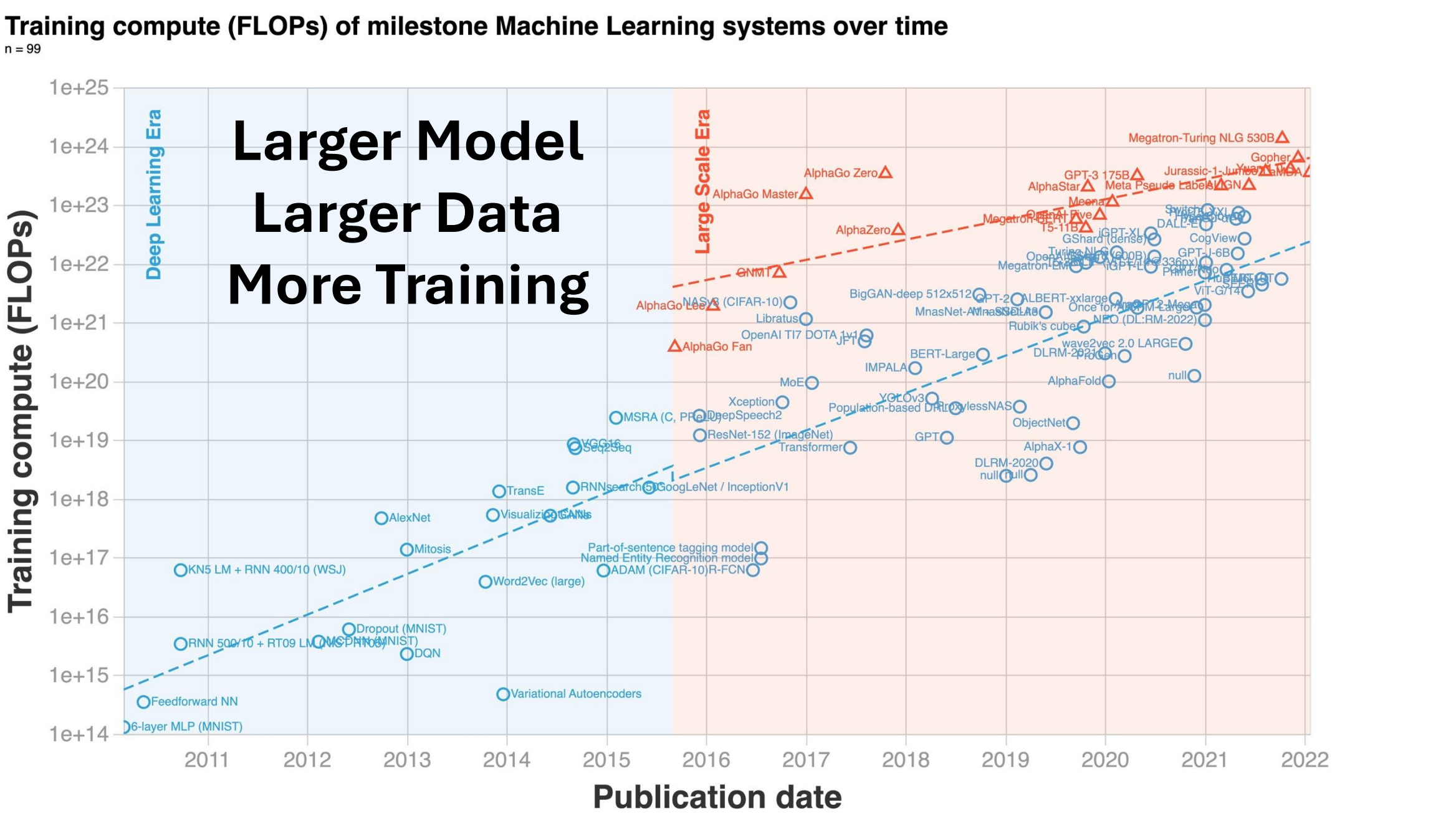
Parallel Training

Chenwei Xu

Northwestern University

Part 1: Overview
Part 2: Data Parallel
Part 3: Model Parallel
Part 4: Mixed Data and Model Parallel
Part 5: Zero-1,2,3
Part 6: DeepSpeed & Huggingface API

Part 1: Overview



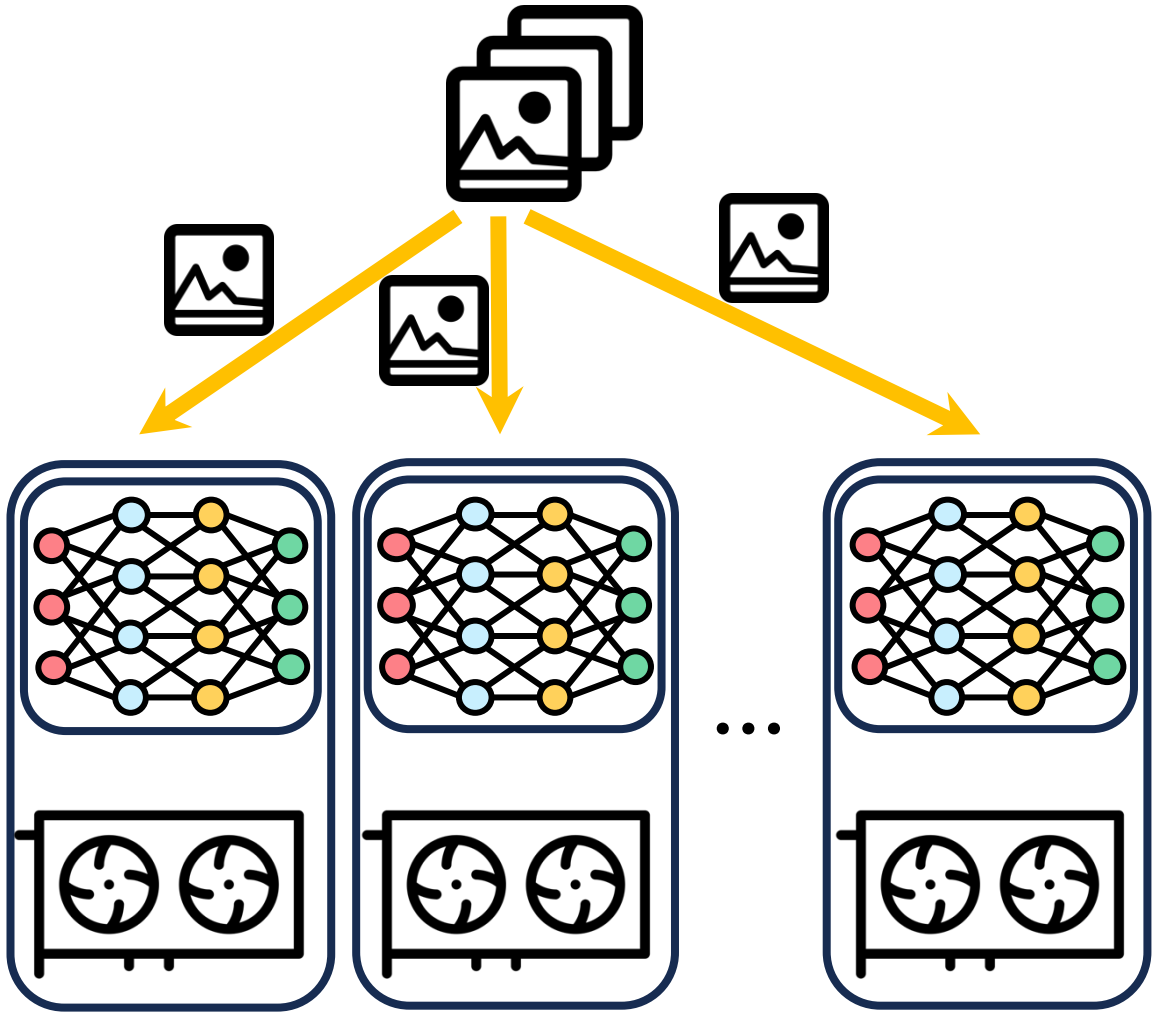
*FLOPs: Floating Point Operations

Parallel Training:
Enable more trainings for larger model and larger data using multiple machines

How?

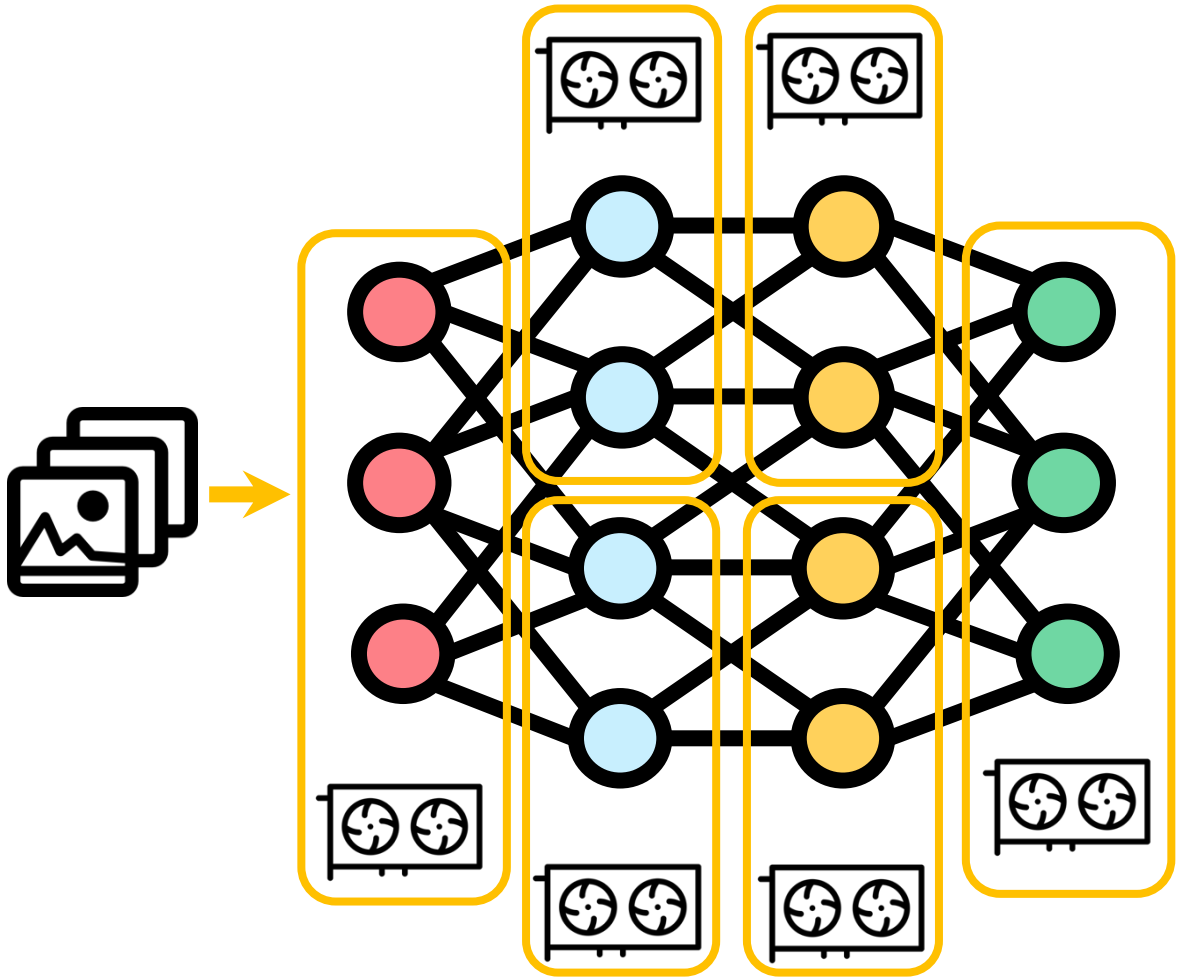
Data Parallel

Partition of Data

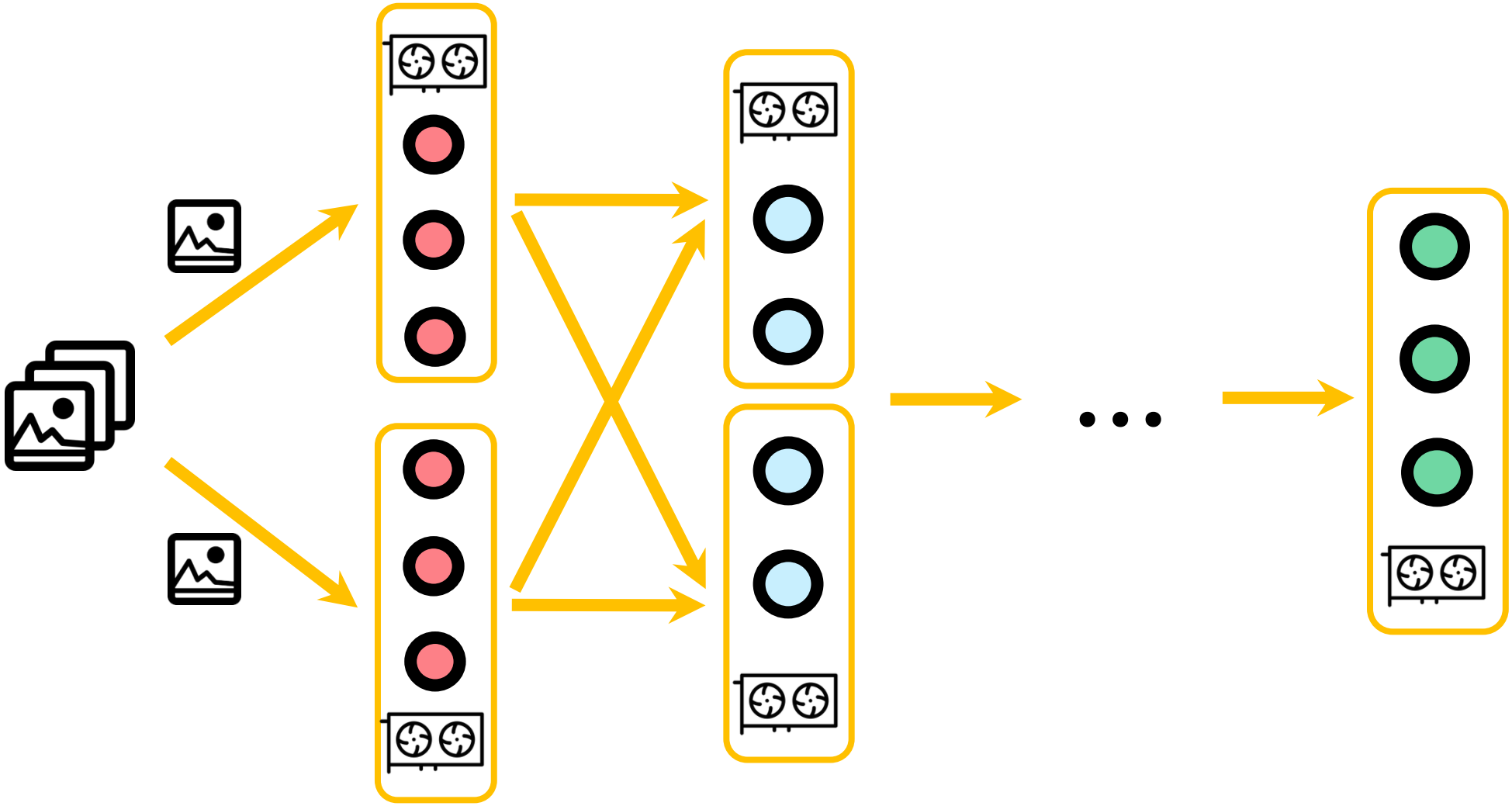


Model Parallel

Partition of Model

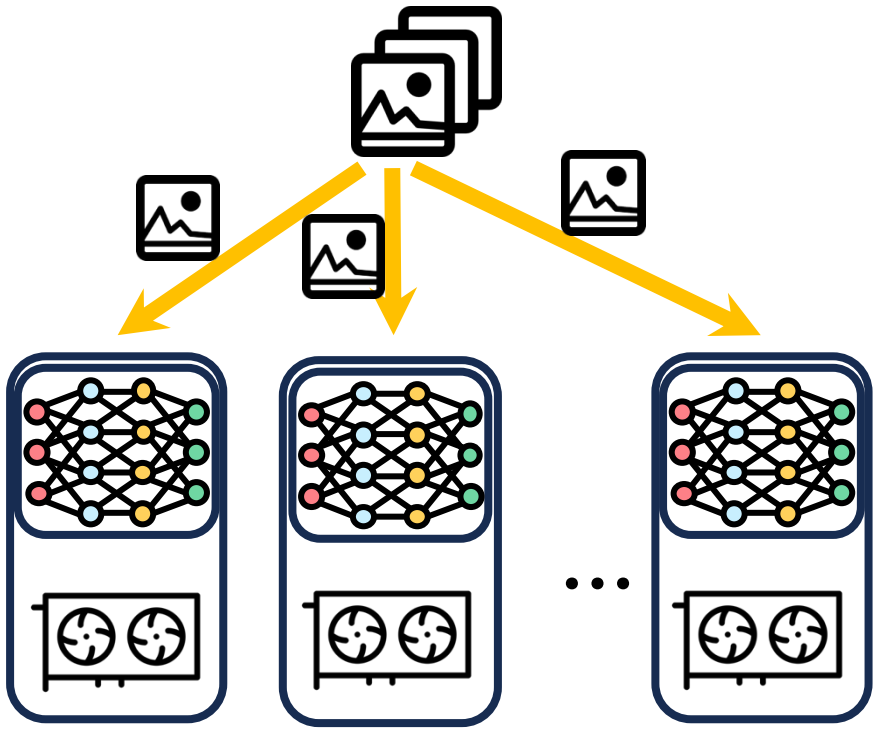


Combined Data and Model Parallel



Part 2: Data Parallel

What is Data Parallel ?

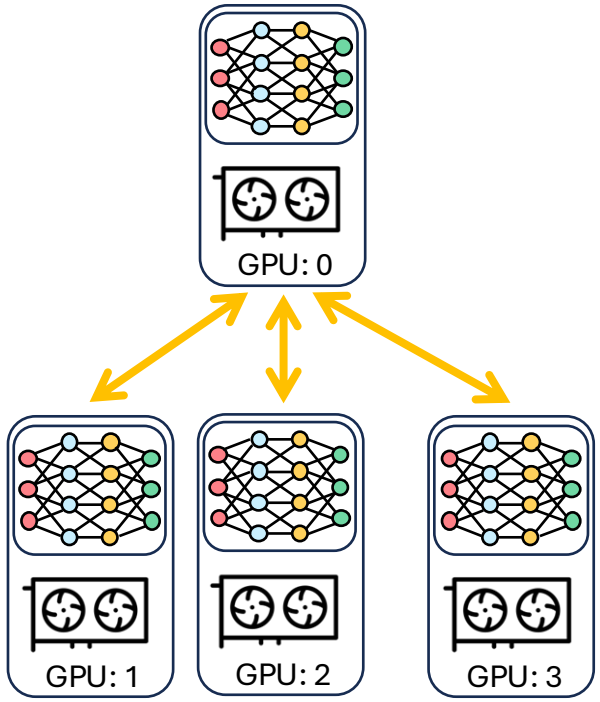


Each GPU:

- holds a **complete** copy of model
- handles **different** batch of data
- computes gradient **independently**

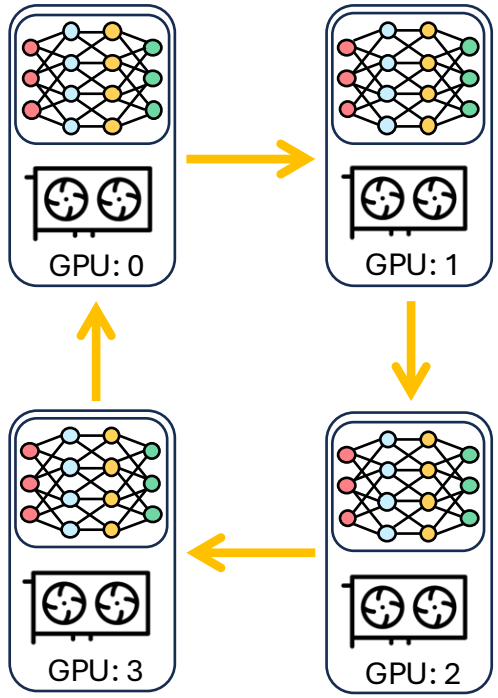
Two Data Parallel Methods:

Naïve Data Parallel



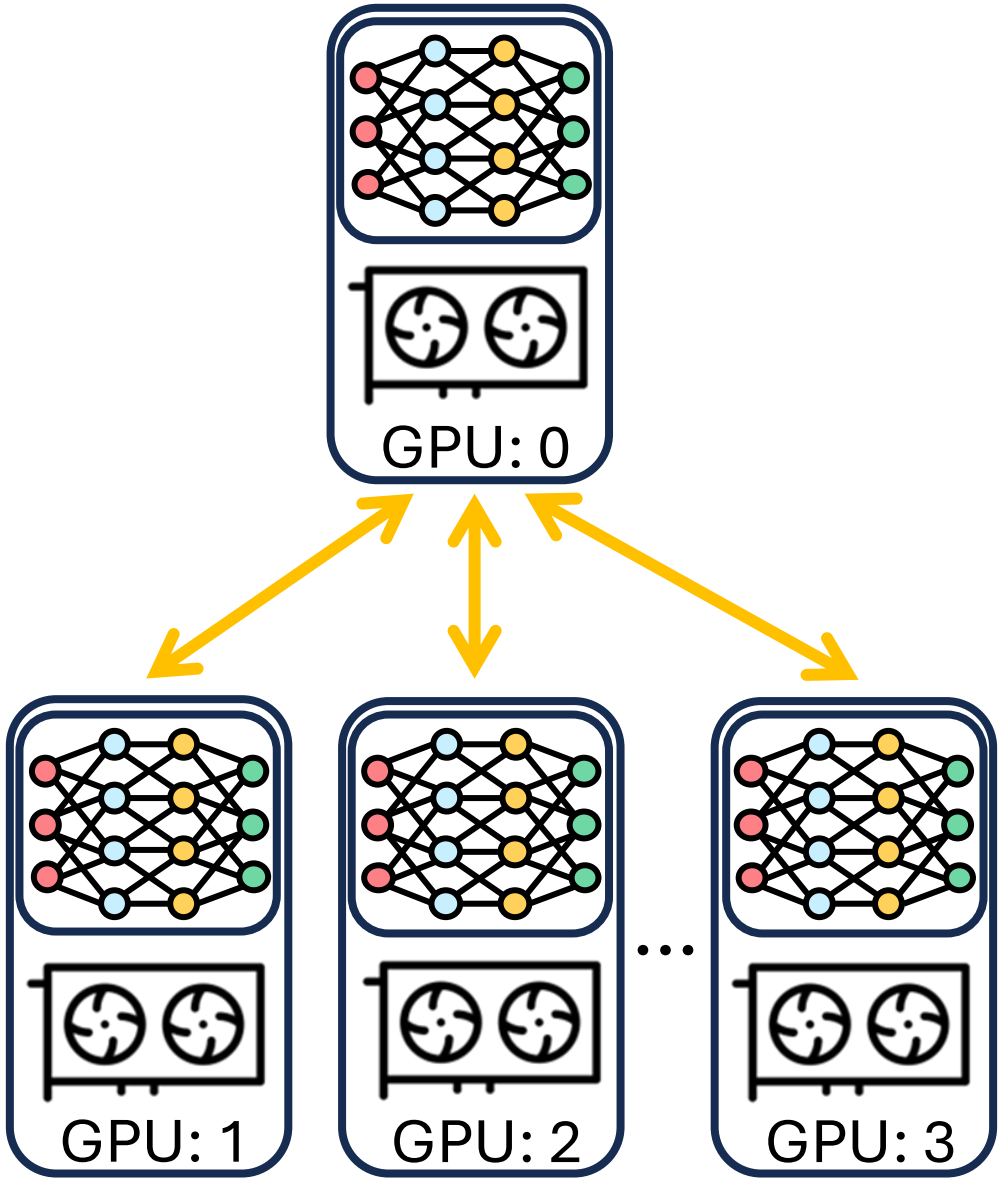
Have main device (GPU :0)

Distributed Data Parallel



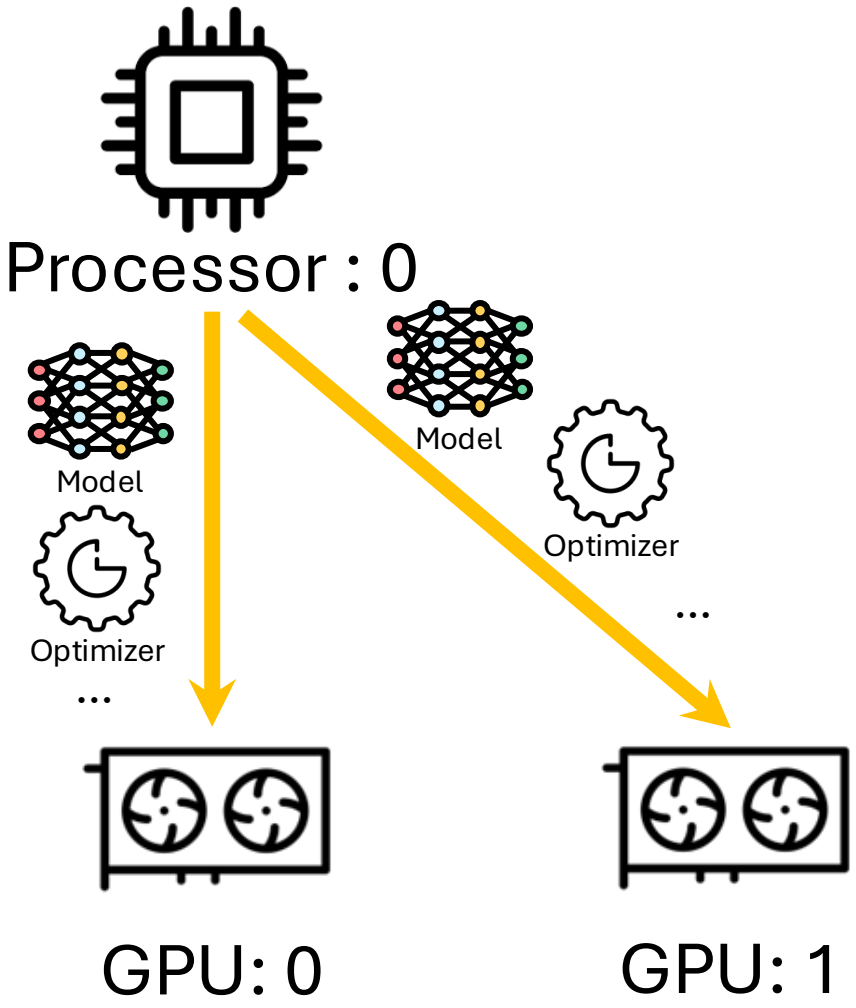
No main device

Naïve Data Parallel



Main device (GPU: 0) communicates to others

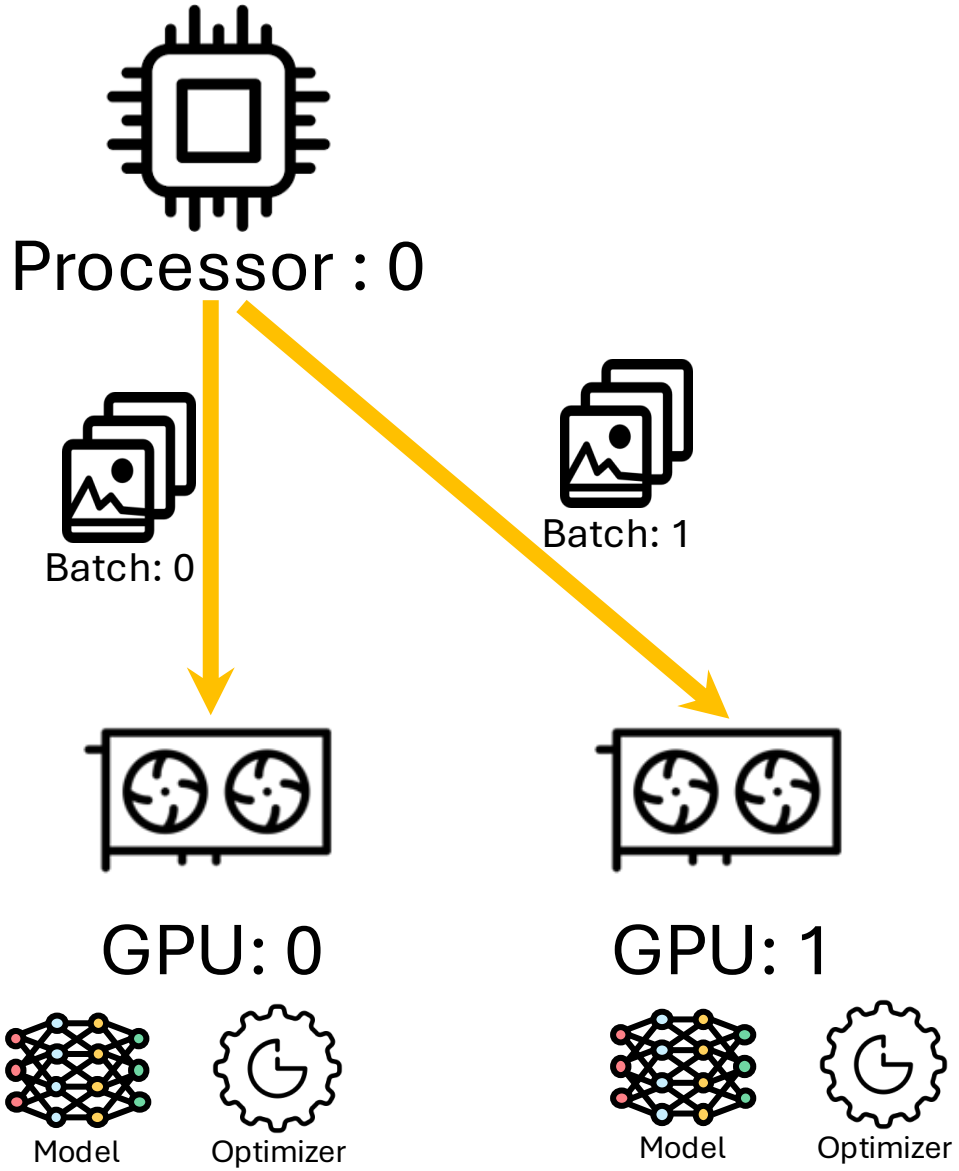
Initialization



Initialization

Single processor sends **model** and **optimizer** to each GPU

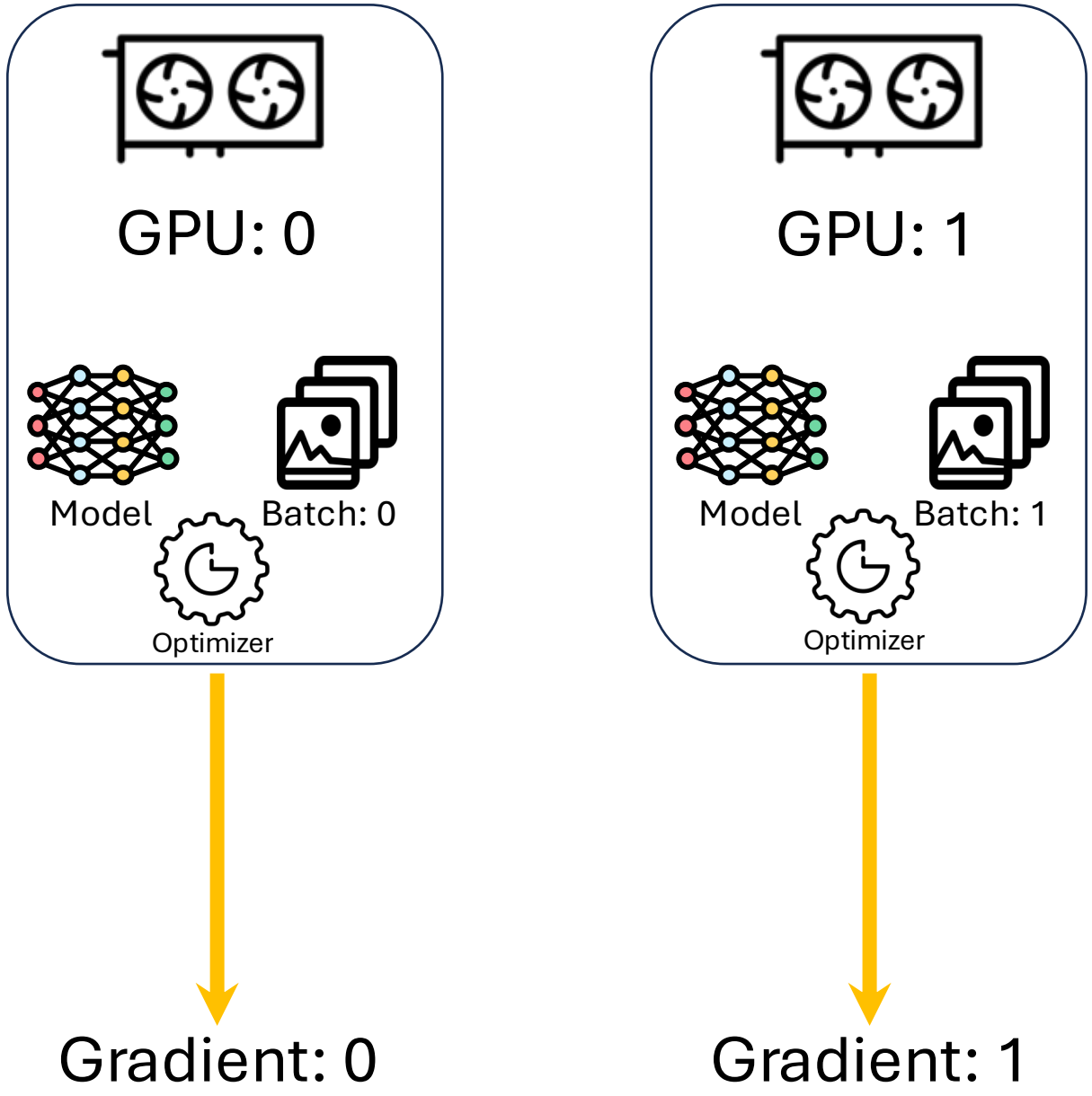
Forward



Forward Pass

Single processor sends **data** to each GPU

Gradient Calculation



Gradient Calculation

Each single GPU calculate gradient independently

Gradient Collection and Aggregation

GPU: 0

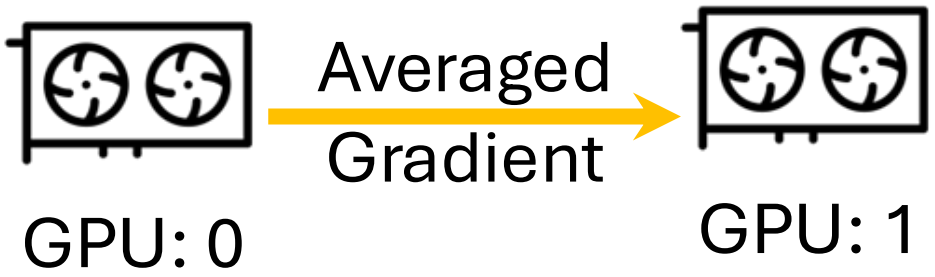
GPU: 1

Average(Gradient: 0, Gradient: 1)

Average Gradient

Main device (GPU: 0) **collect** all gradients to the and **average**

Gradient Distribution



Gradient Distribution

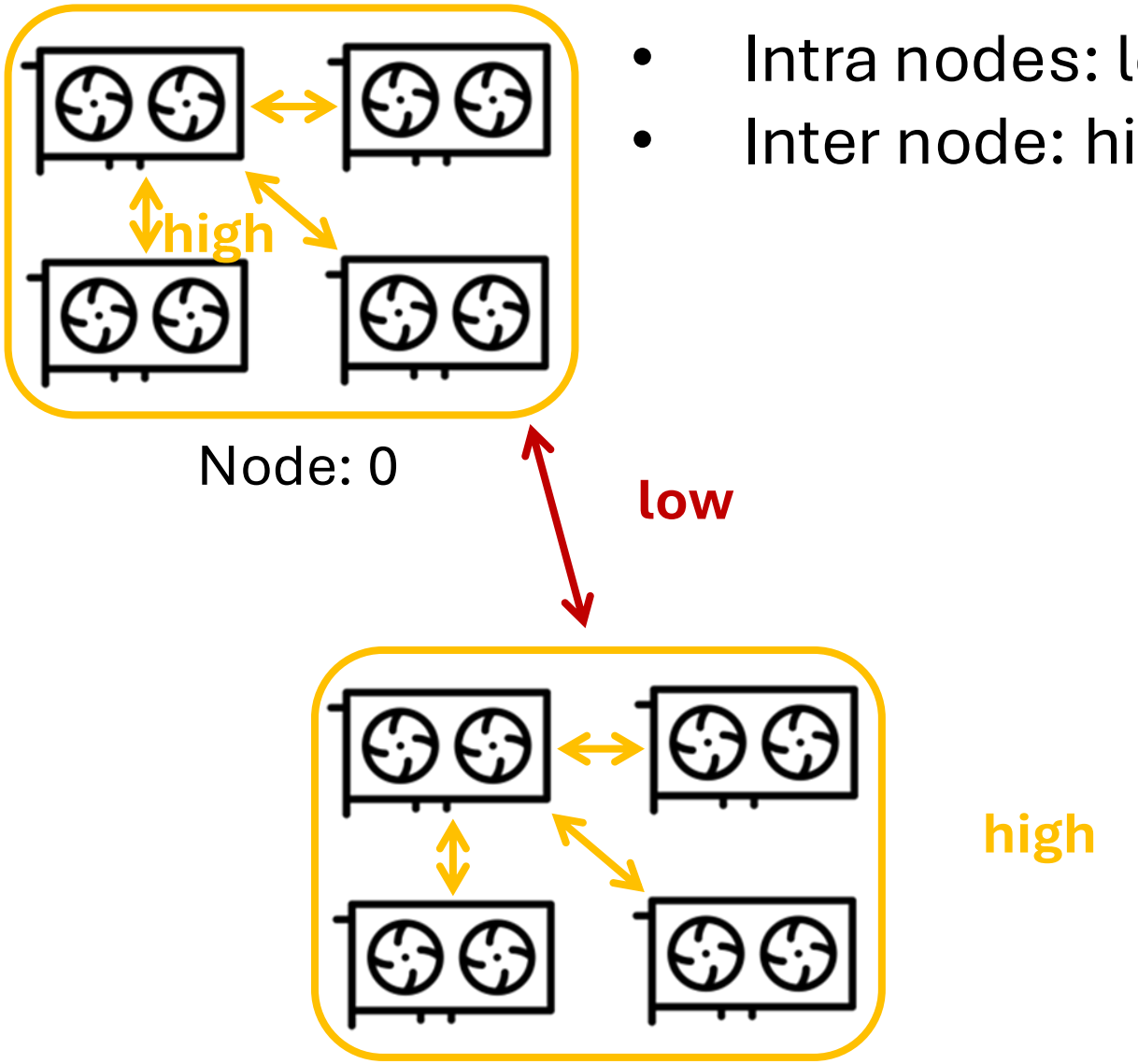
GPU:0 **send** averaged gradient to each other devices

Model Update

Each GPU updates its parameter use averaged gradient

Communication:

- Intra nodes: low bandwidth
- Inter node: high bandwidth

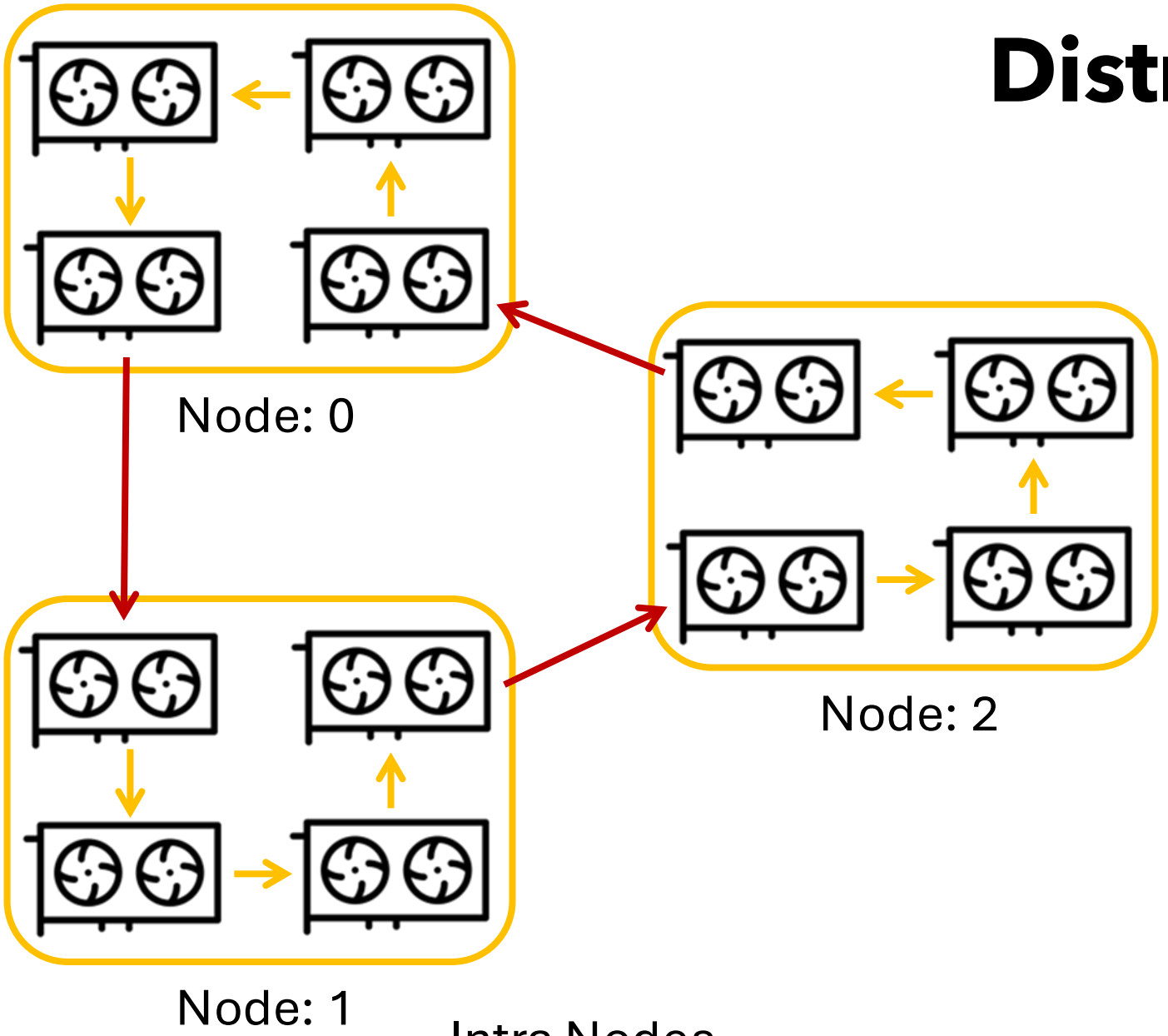


Limitation:

- Only favor single node* setting

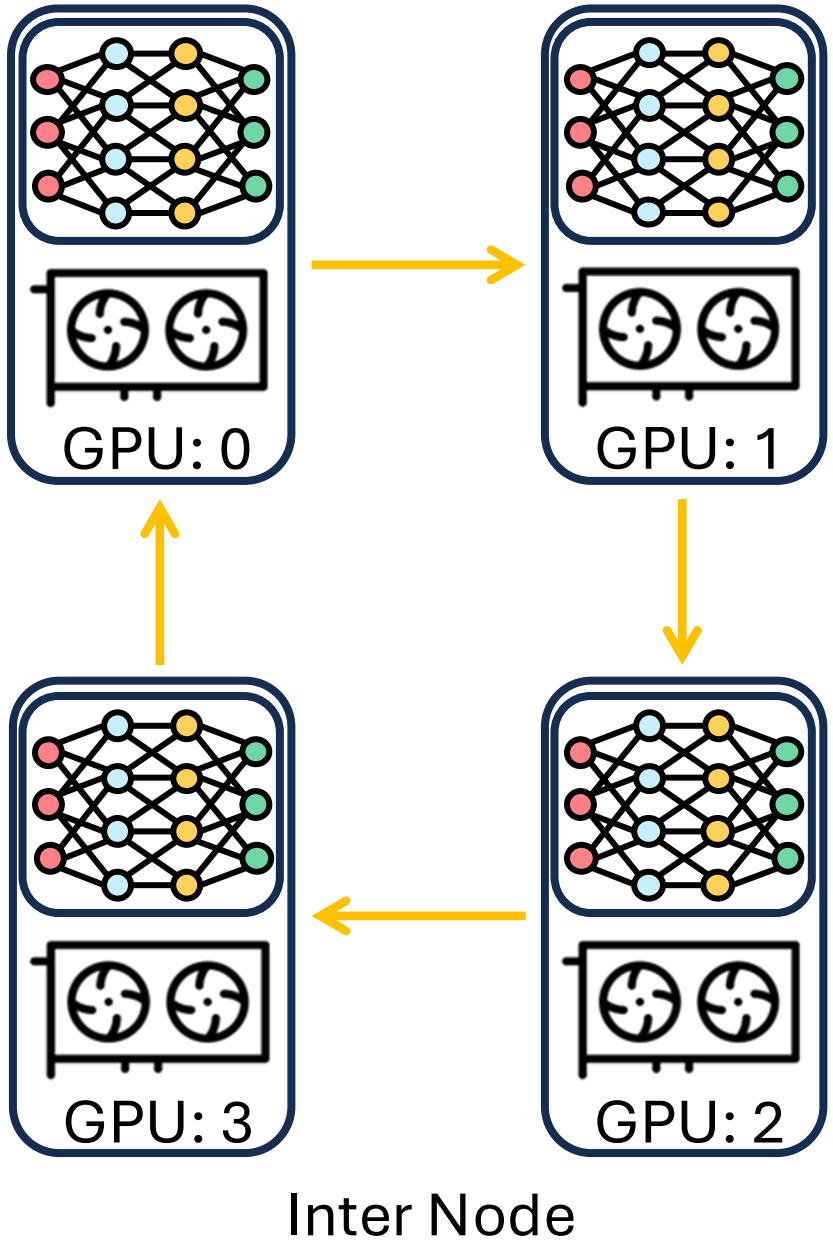
node*: a single computation unit in cluster* contains multiple GPUs
Cluster*: a server contains multiple nodes
bandwidth*: number of data communicate per unit of time

Distributed Data Parallel (DDP)



Enables multiple nodes training

PyTorch Distributed: Experiences on Accelerating Data Parallel Training



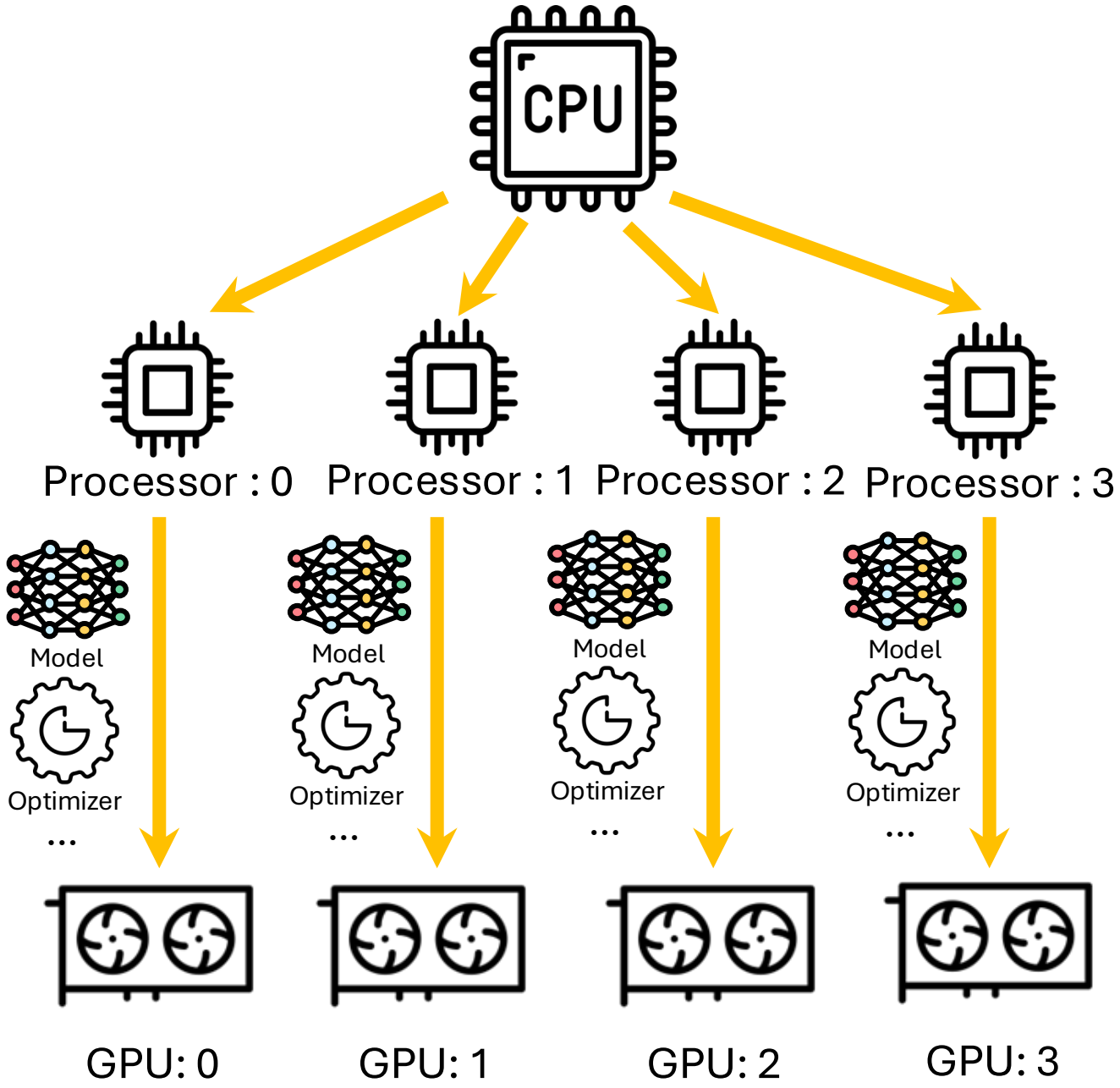
Distributed Data Parallel (DDP)

Each device/node communicates to the other one in a **“Ring”**

No “**main device**” in this scenario

Distributed Initialization

Inter Node

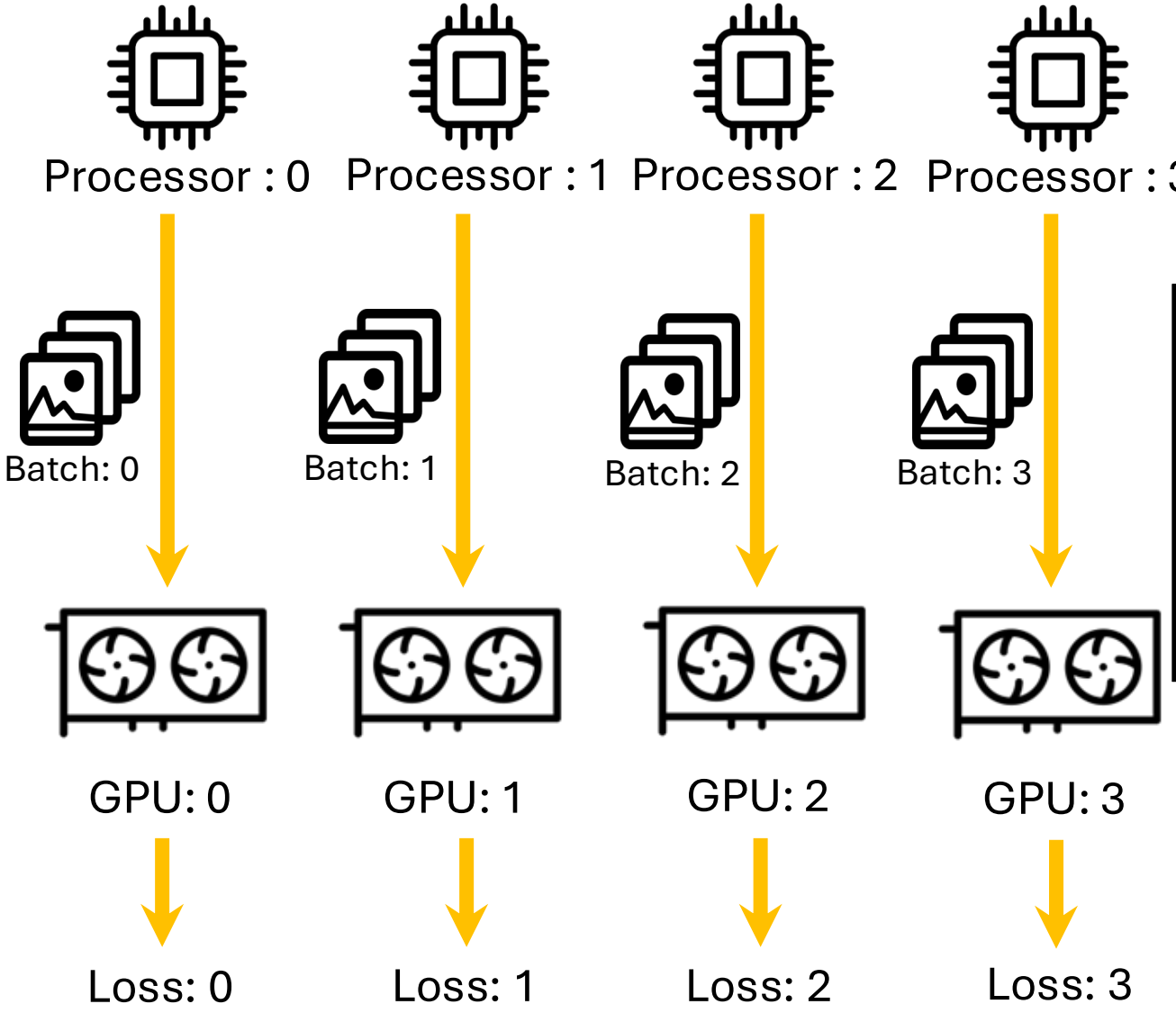


Initialization

Each processor sends **model** and **optimizer** to each GPU

Forward

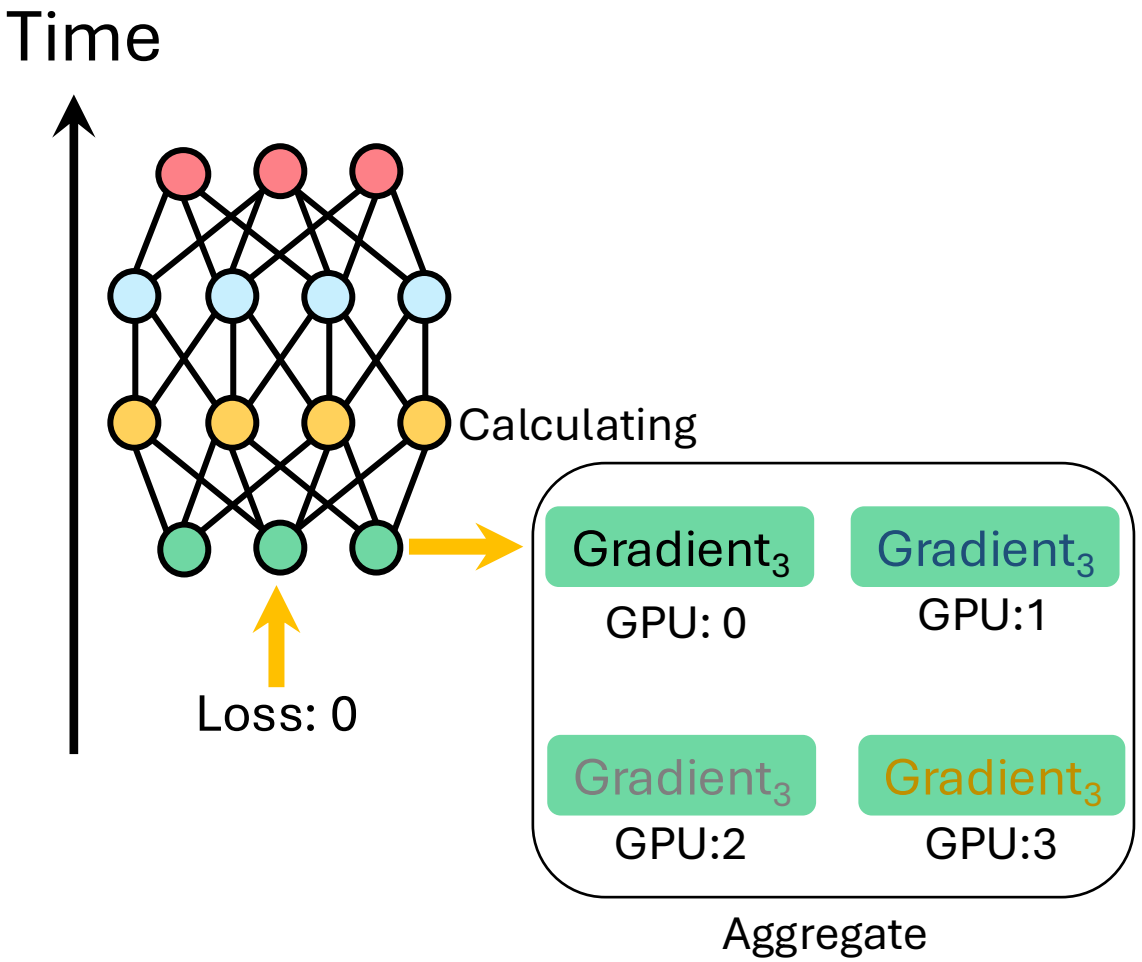
Inter Node



Forward Pass

Each processor sends **data** to each GPU, calculates loss independently

DDP Gradient Update



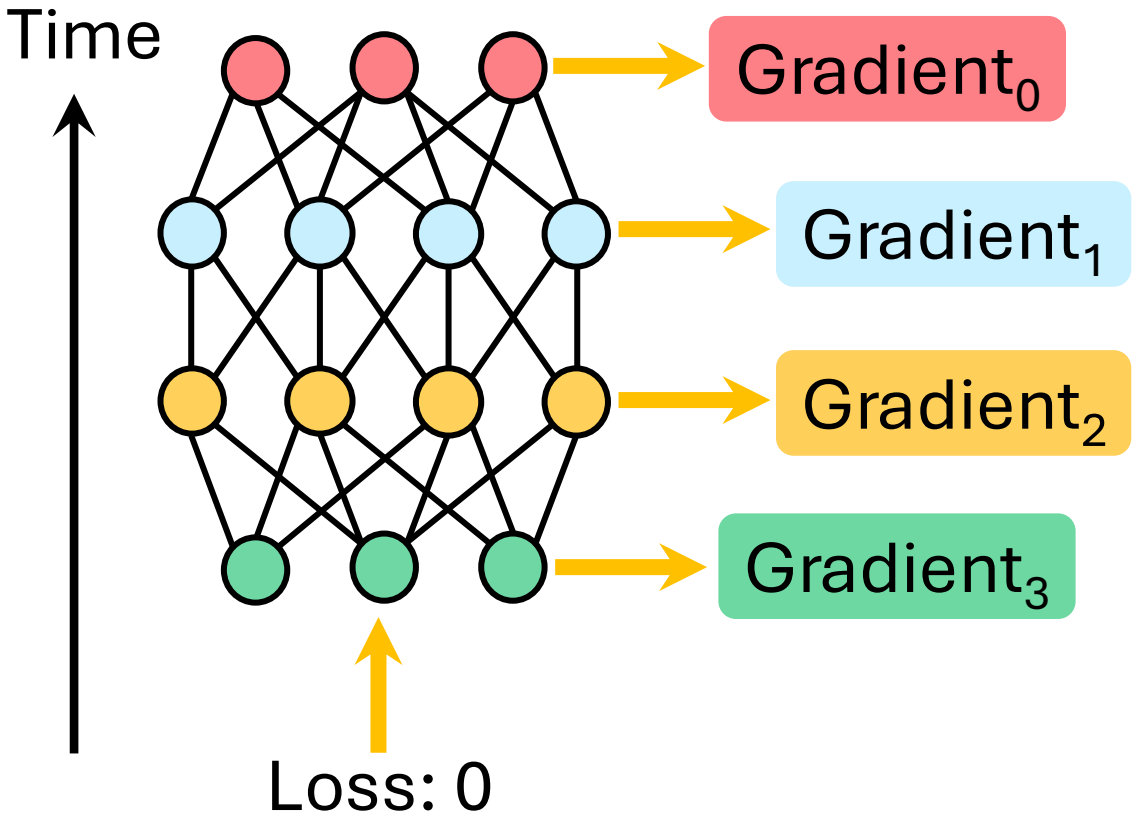
Intuition

Calculate and Communicate at the **same time**

Aggregate **small** amounts of gradients one time

Gradient Calculation

Backpropagation

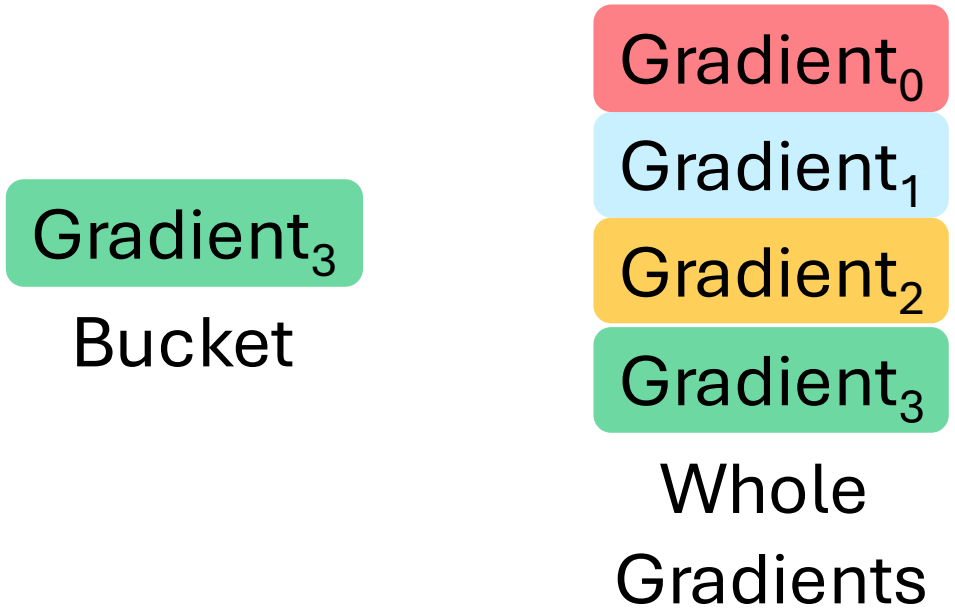


Tow Observations

Gradients are calculated **sequentially**

Each layer's gradients are **much smaller** than whole model gradients

Gradient Buckets



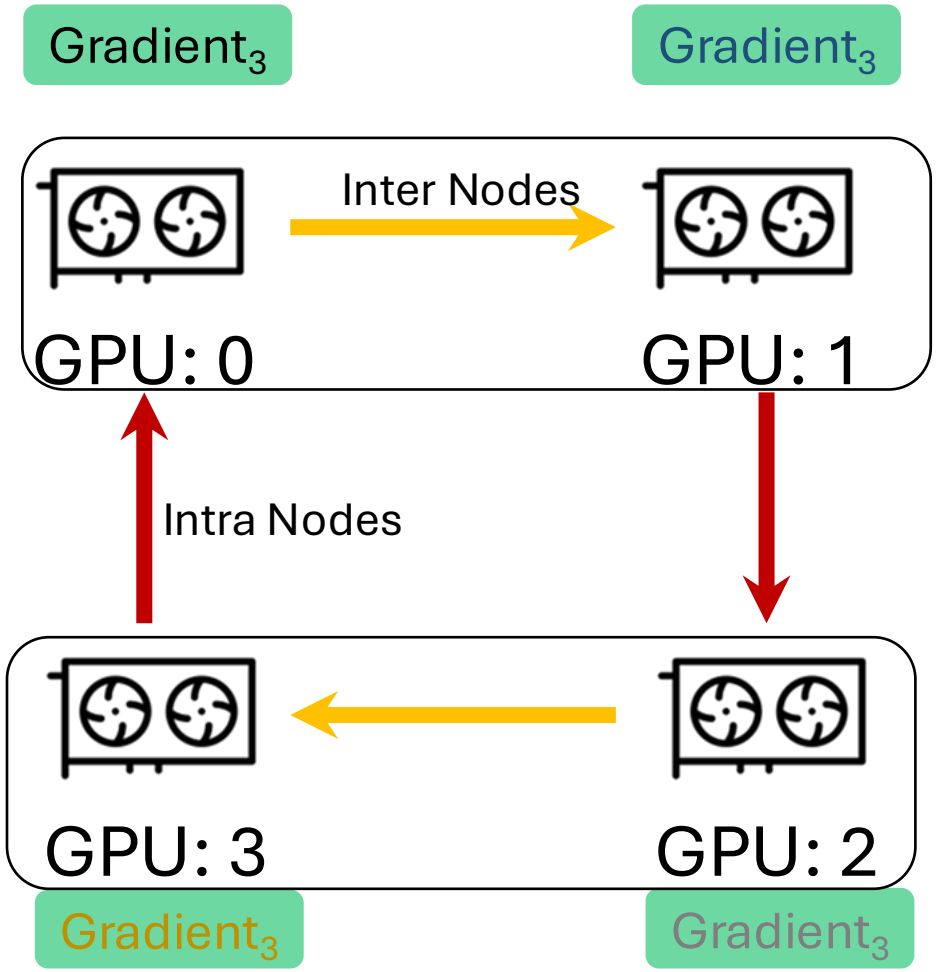
Bucket

A small portion of gradients to communicate

Each communication only about one bucket instead of whole model, why?

Communication Overhead

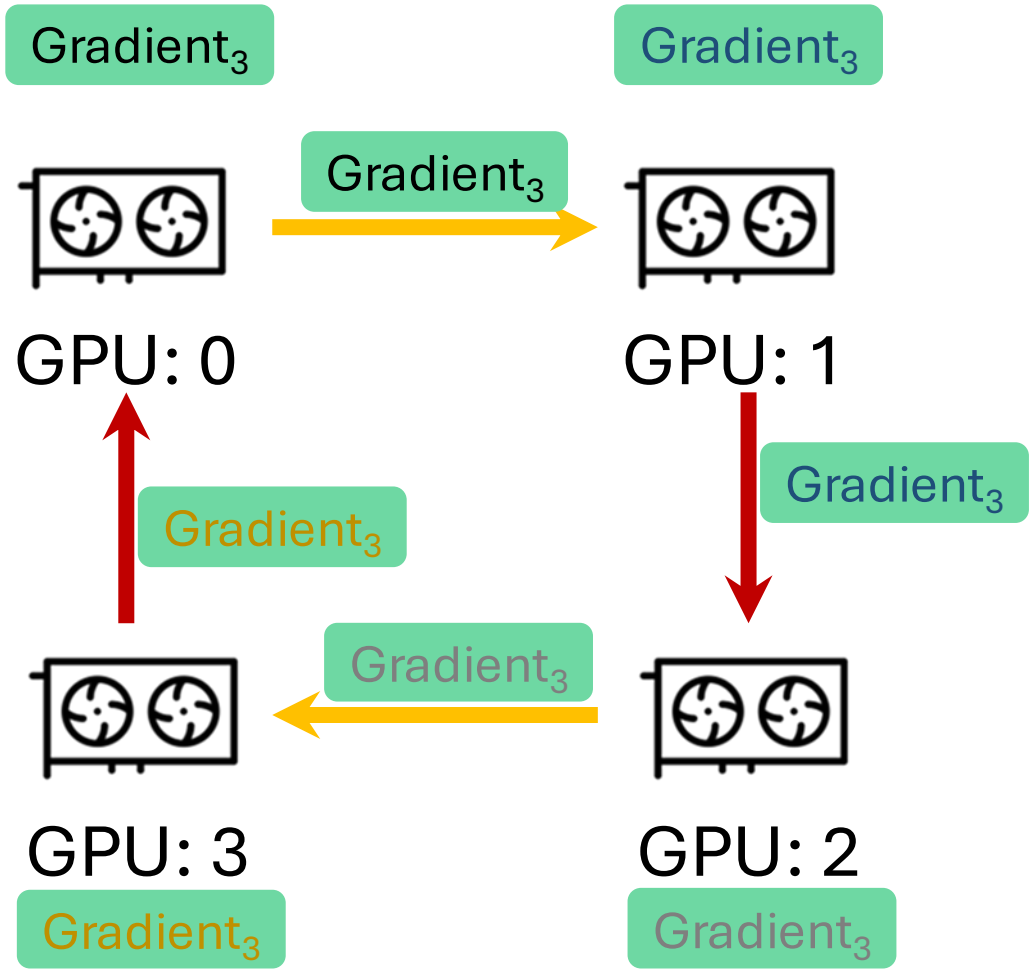
How to Send and Recieve Buckets?



DDP

Each GPU only communicates to adjacent one

Gradient Communication



(n-1 communication total)

n: number of total GPUs

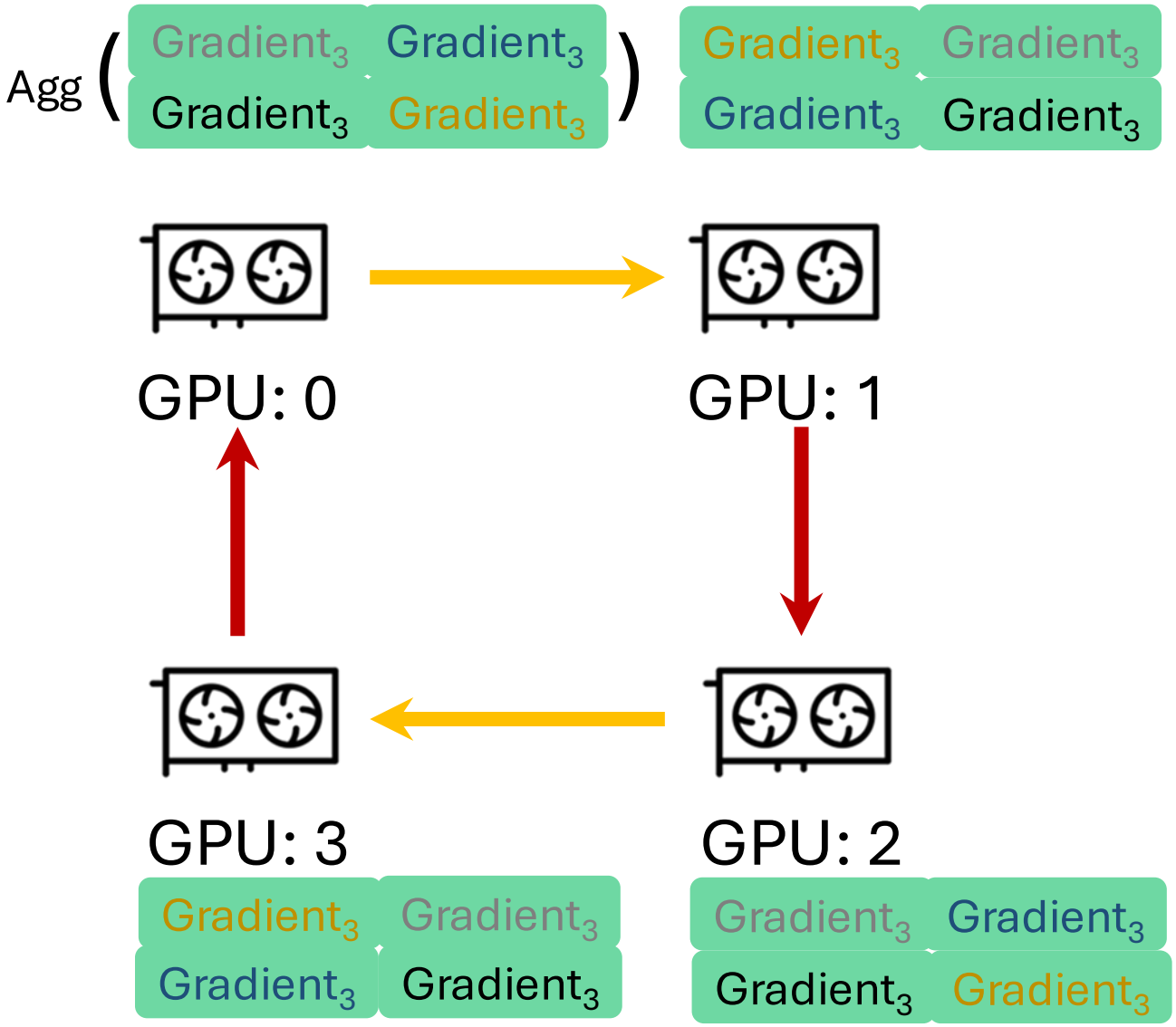
Ring Communication

For each GPU:

Send its gradient to next GPU

Receive previous GPU's gradient

Gradient Average

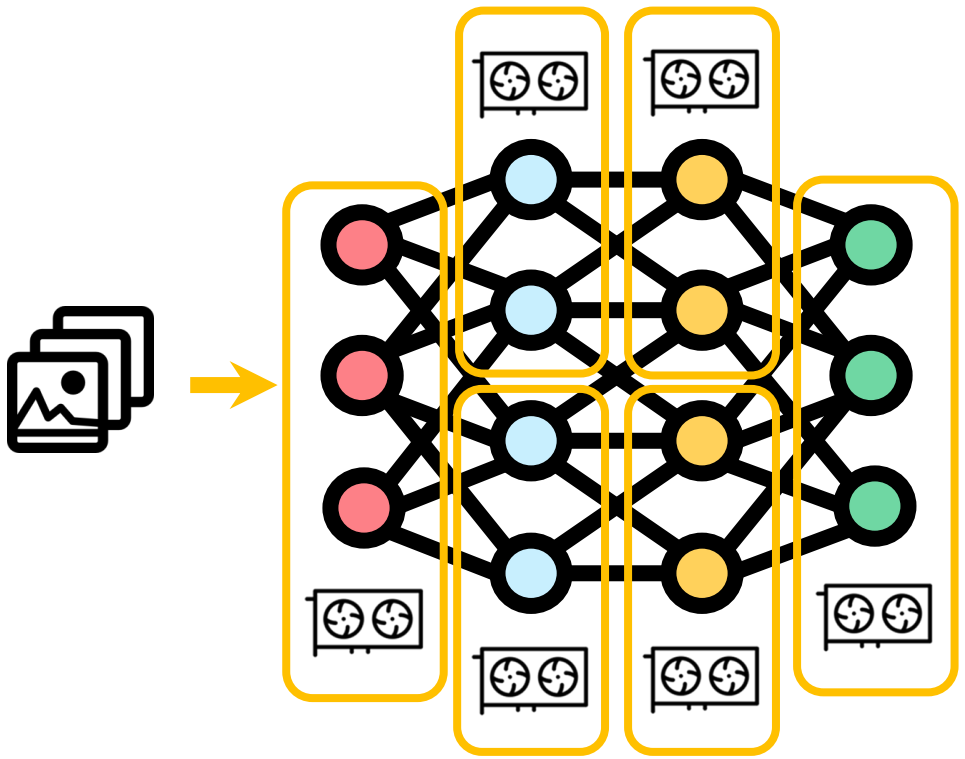


Ring AllReduce:

Aggregate gradients on each machine once receive all gradients

Part 3: Model Parallel

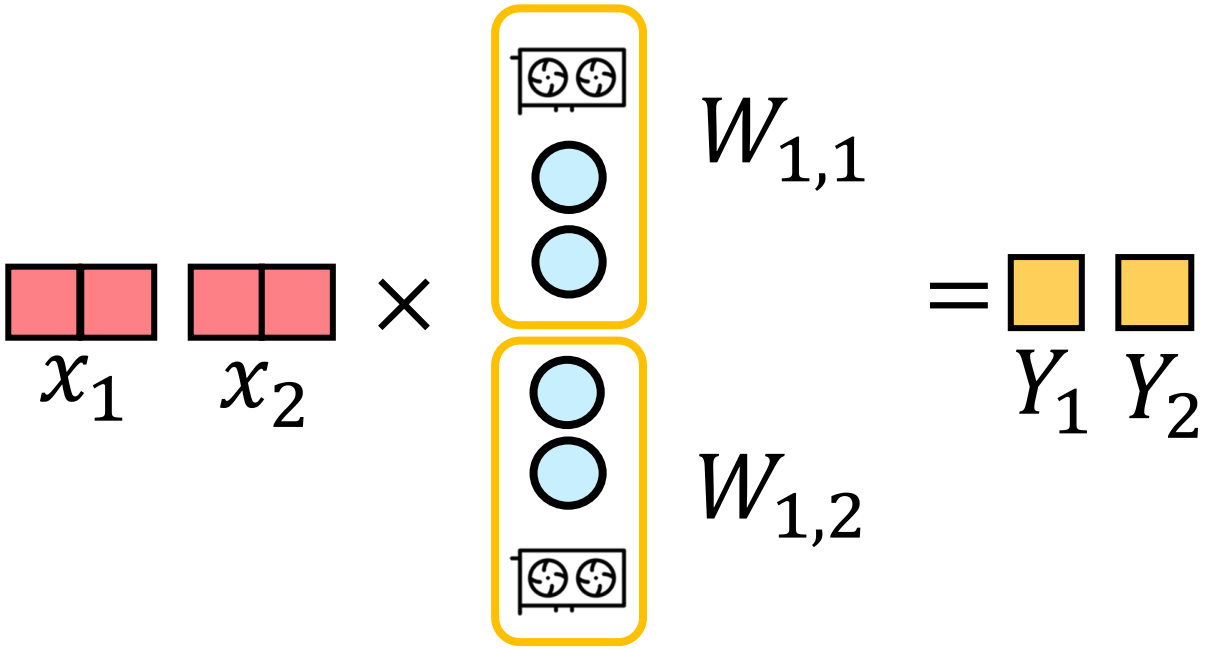
What is Model Parallel ?



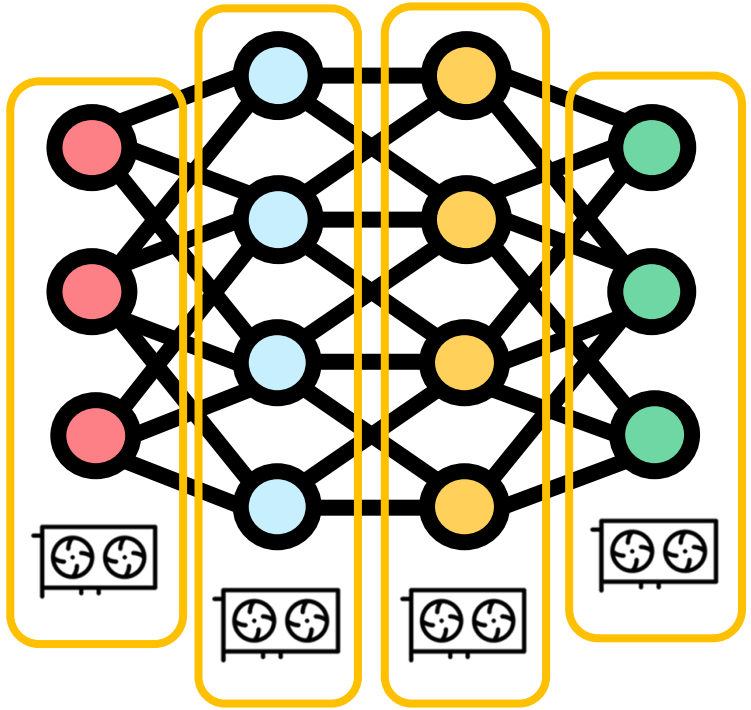
- Each GPU holds a **part** of model
- Each GPU computes **dependently**

Two Model Parallel Methods:

Tensor Parallel*



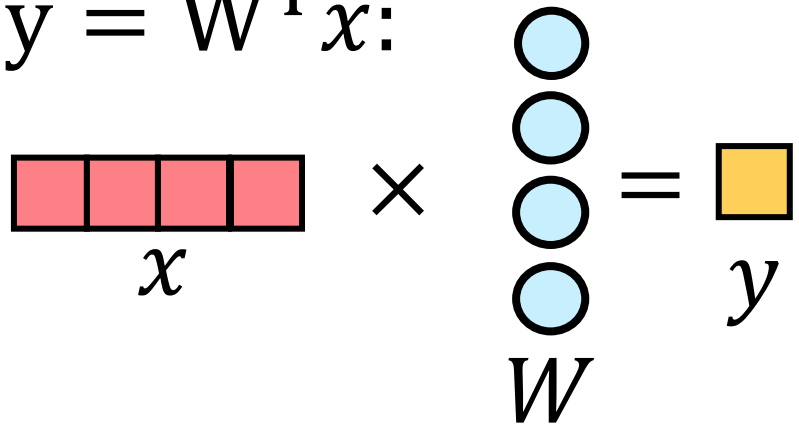
Pipeline Parallel



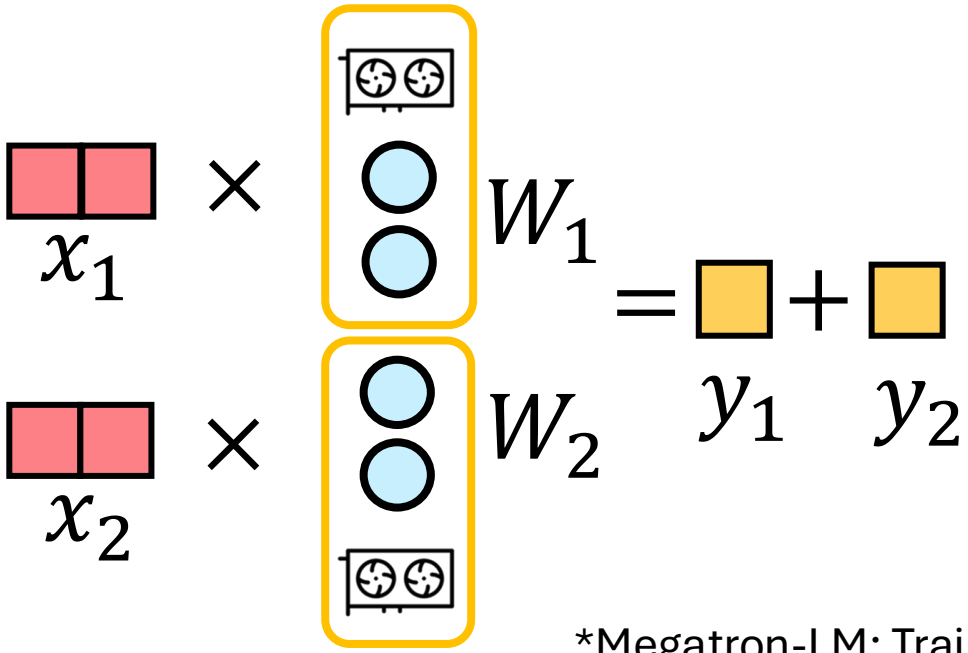
* tensor manipulation: mathematical operations like addition, multiplication

Tensor Parallel

$y = W^T x$:



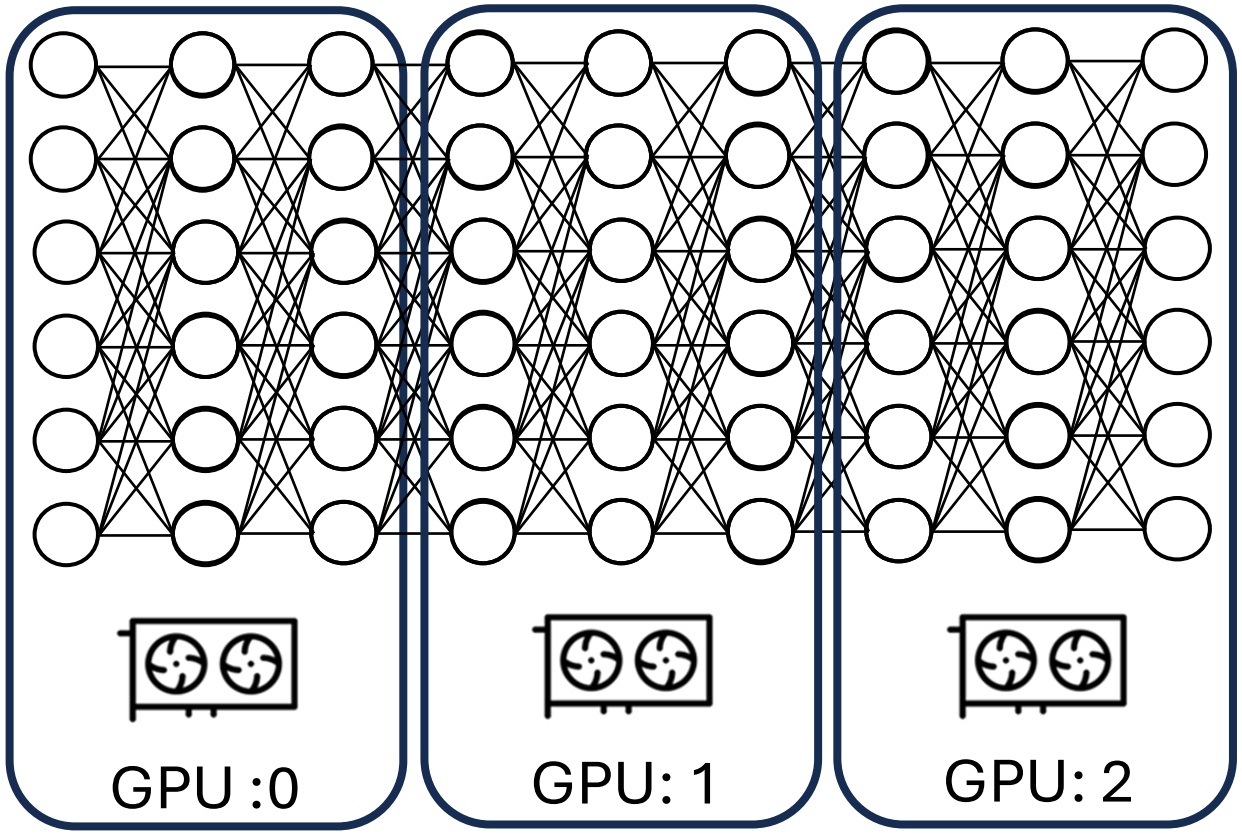
Tensor Parallel:



- Split tensor manipulation in different machine

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

Pipeline Parallel (Naïve*)



- Split model layers in different parts
- Each part in a single GPU

* Advanced Approach in Part 4: Mixed Parallel

When use?

If single GPU cannot hold single tensor manipulation*:

Tensor Parallel

Else if single GPU cannot hold whole model:

Model Parallel

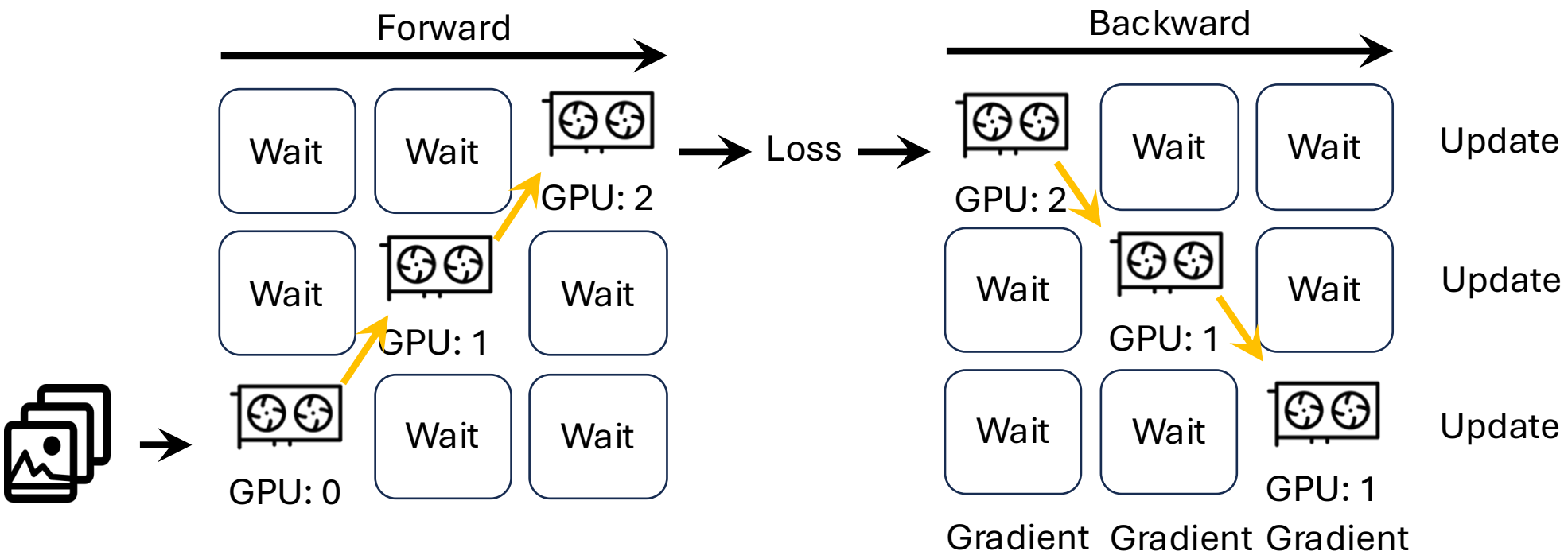
Else:

Data Parallel

* tensor manipulation: mathematical operations like addition, multiplication

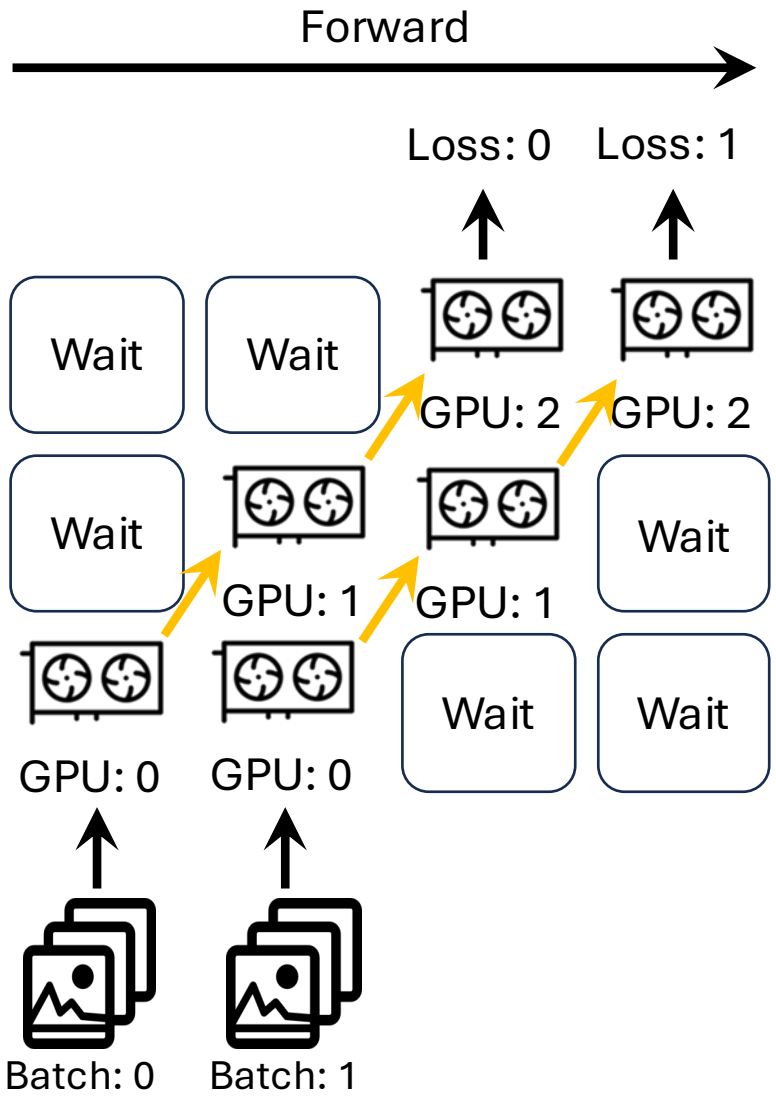
Part 4: Mixed Parallel

Key Issue in Pipeline Parallel: GPU Waiting



Only 1 GPU works each moment

Solution for Previous Issue

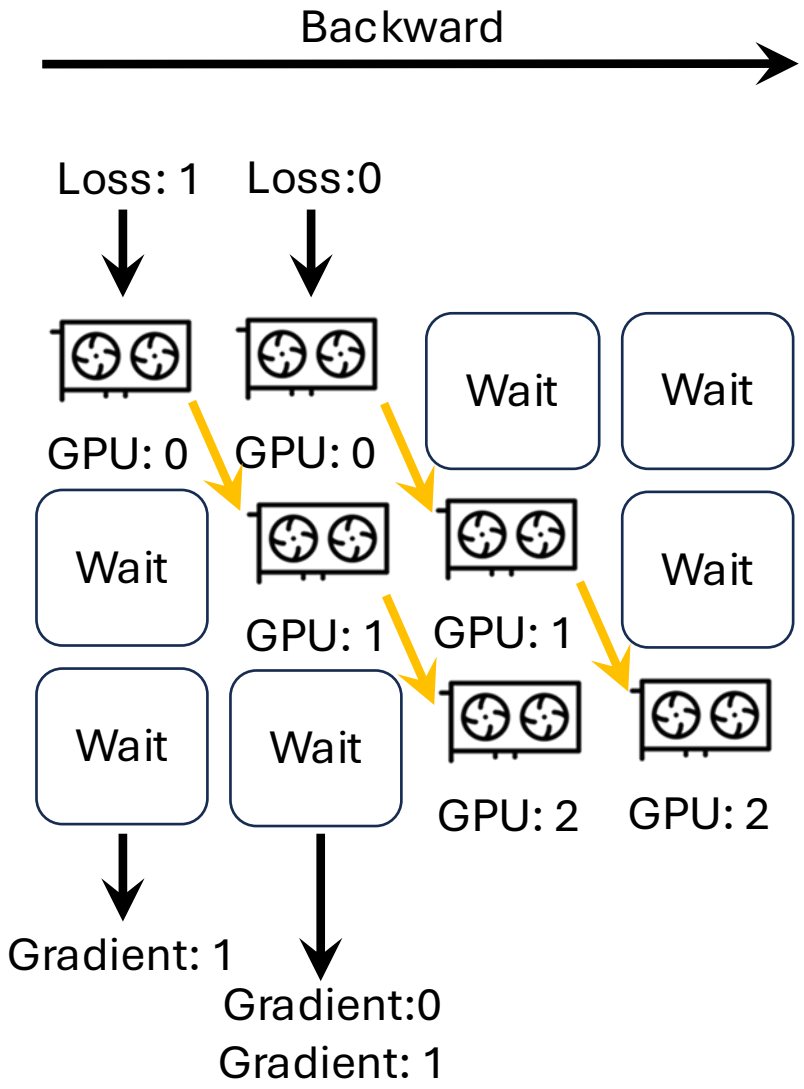


+Data Parallel:

When one GPU finished current batch, let it run another batch

*GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Solution for Previous Issue

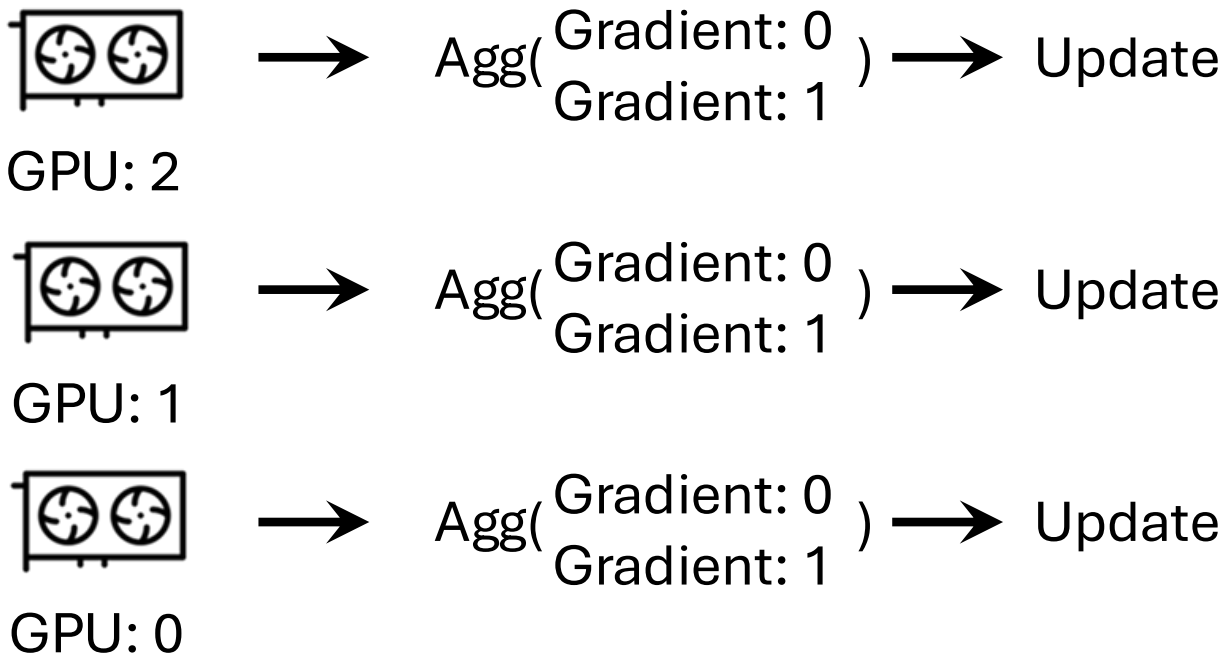


Compute Gradients:

Compute the gradient for each parts

*GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Solution for Previous Issue

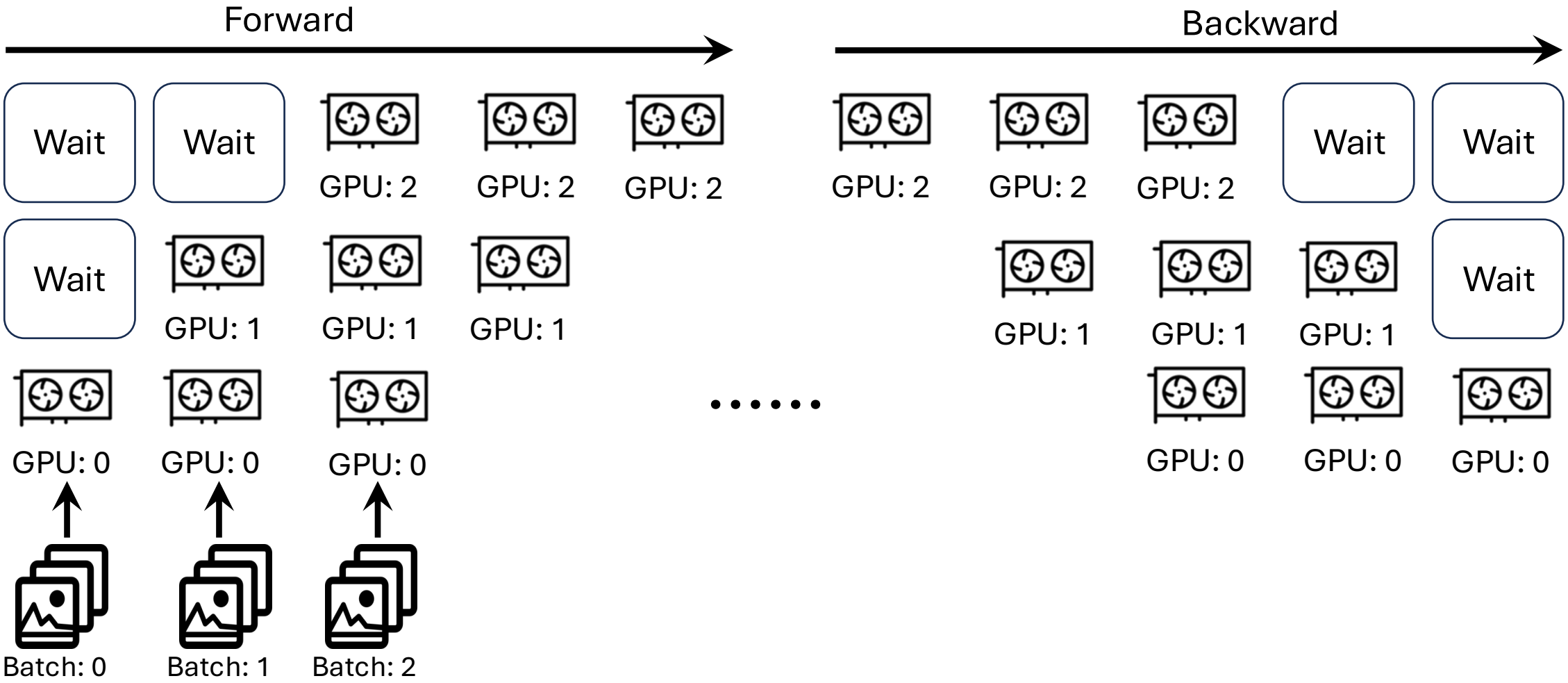


Model Update:

Once one iteration finished, update the model using aggregating gradients

*GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Reduce the Percentage of “Wait” With more Batches

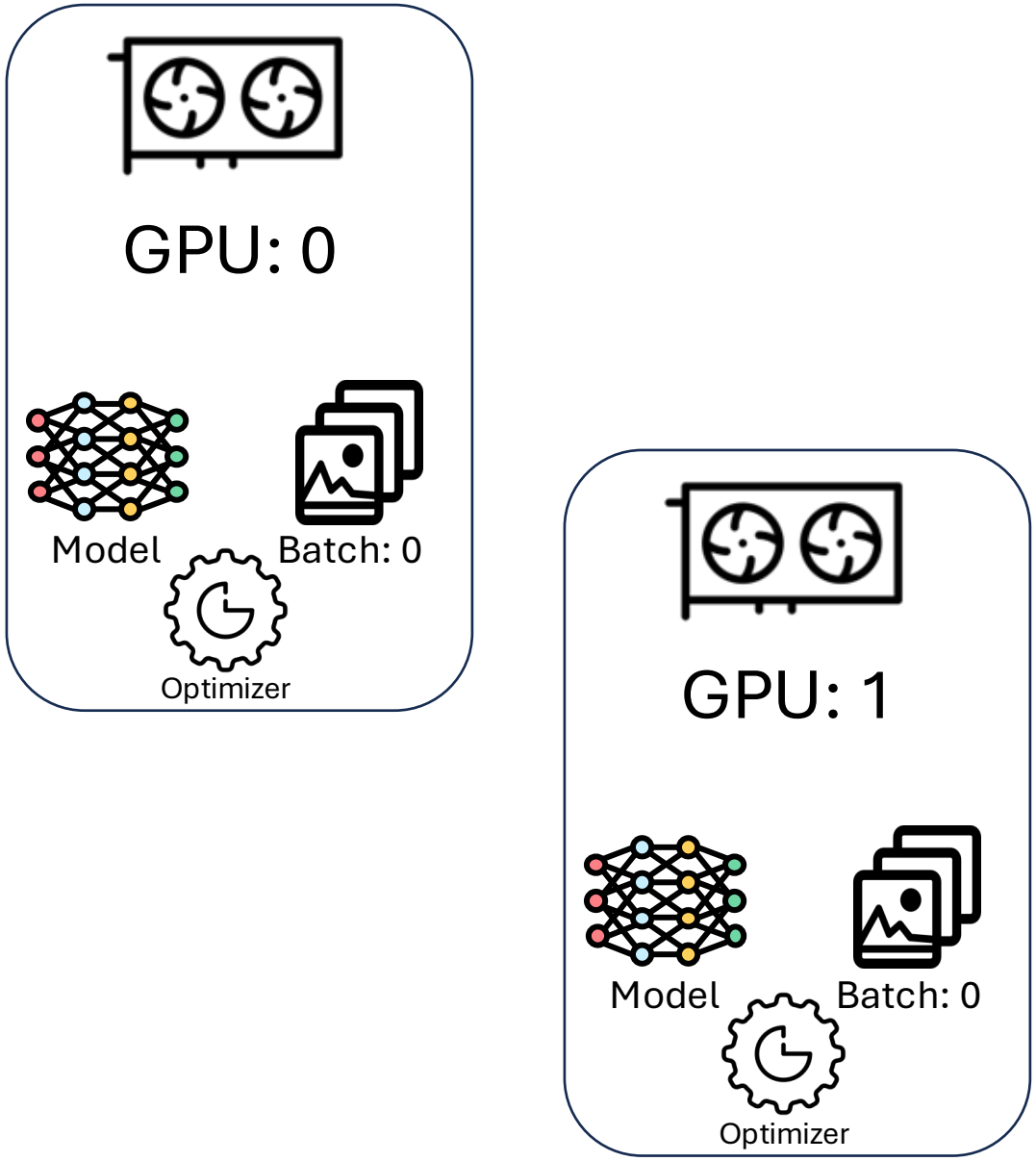


*GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Part 5: Zero-1,2,3

Zero is **Memory Optimization** for Optimizer, Gradients, Parameters

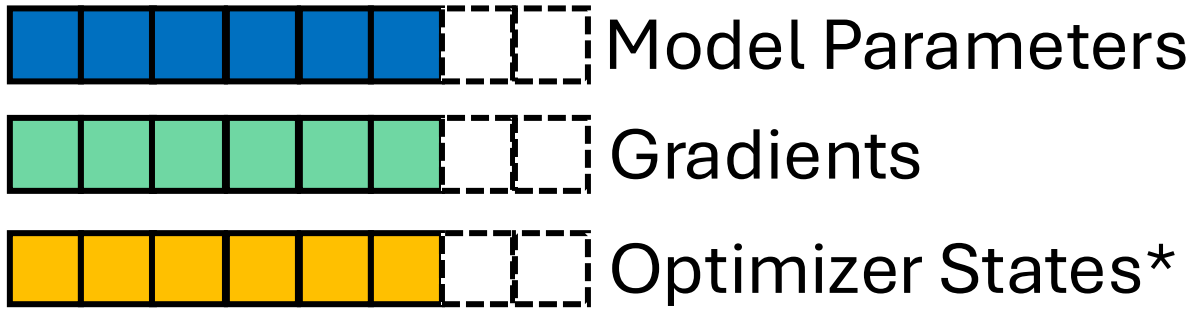
Memory Issue in Data Parallel



Memory Cost:

Each GPU keeps a **whole** copy of Parameters, Gradients, and Optimizer

Larger Model, More Memory Occupancy

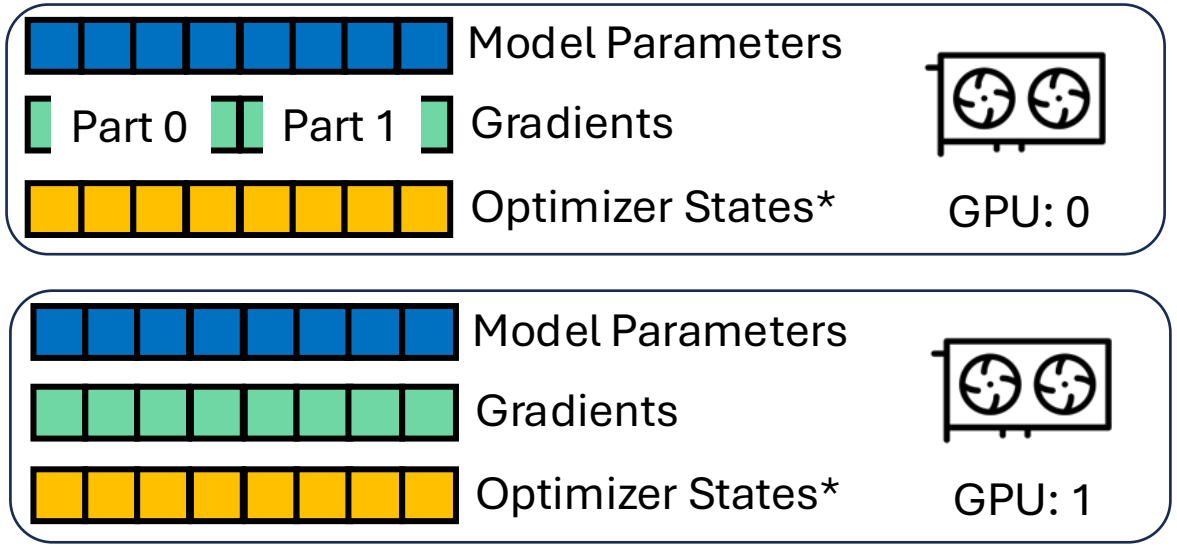


How much memory needs for 1 more parameter?

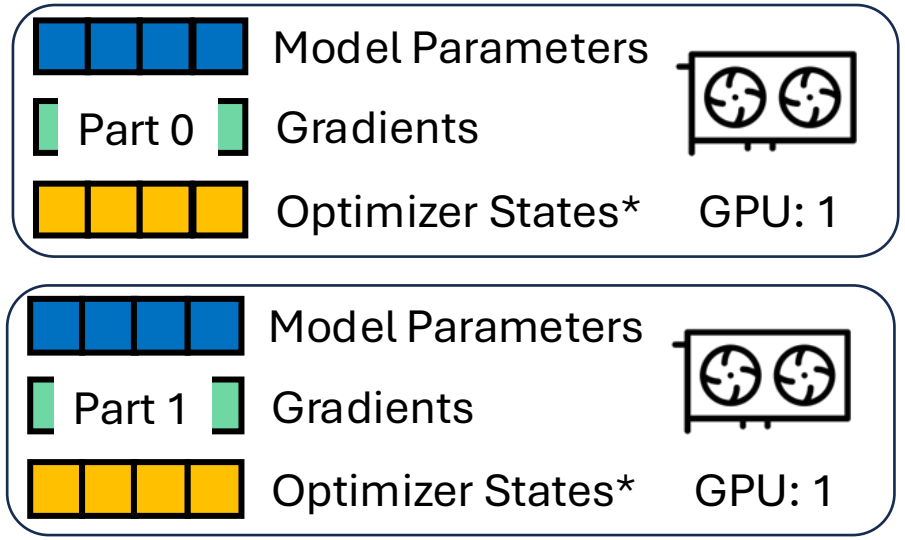
model parameter + gradient + optimizer states*

Optimizer States*:
States needed for updating gradients, such as variance of gradients in Adam

What is Zero?



Data Parallel

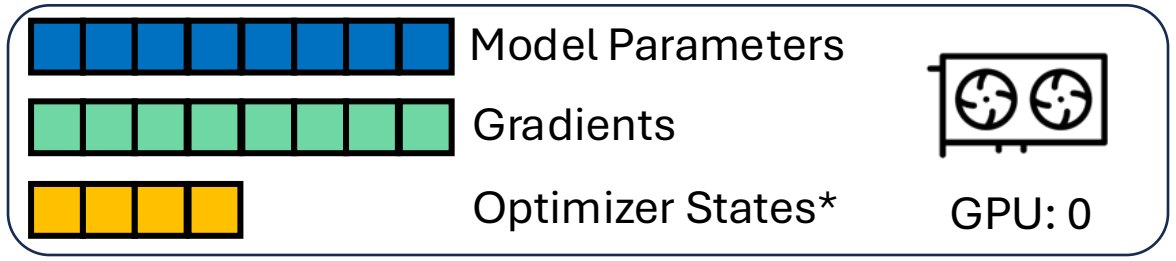


Zero

Reduce Memory Cost:

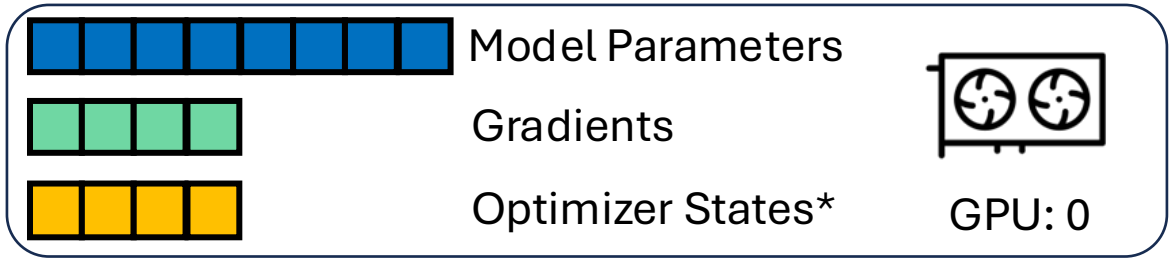
Each GPU keeps a **part** of Parameters, Gradients, and Optimizer

Three Stages of Zero



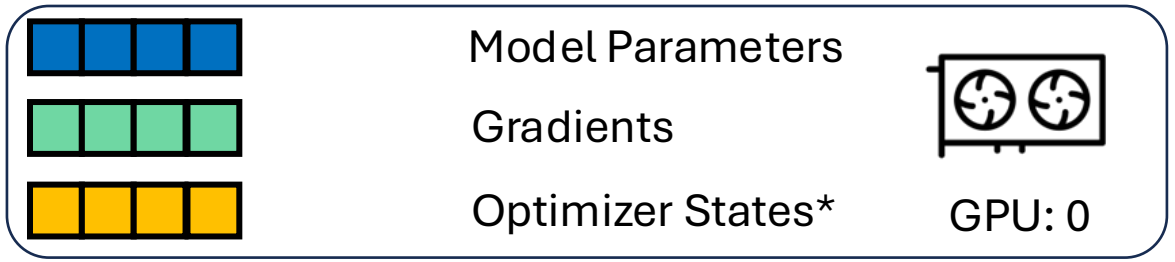
Zero-1

Reduce Optimizer State



Zero-2

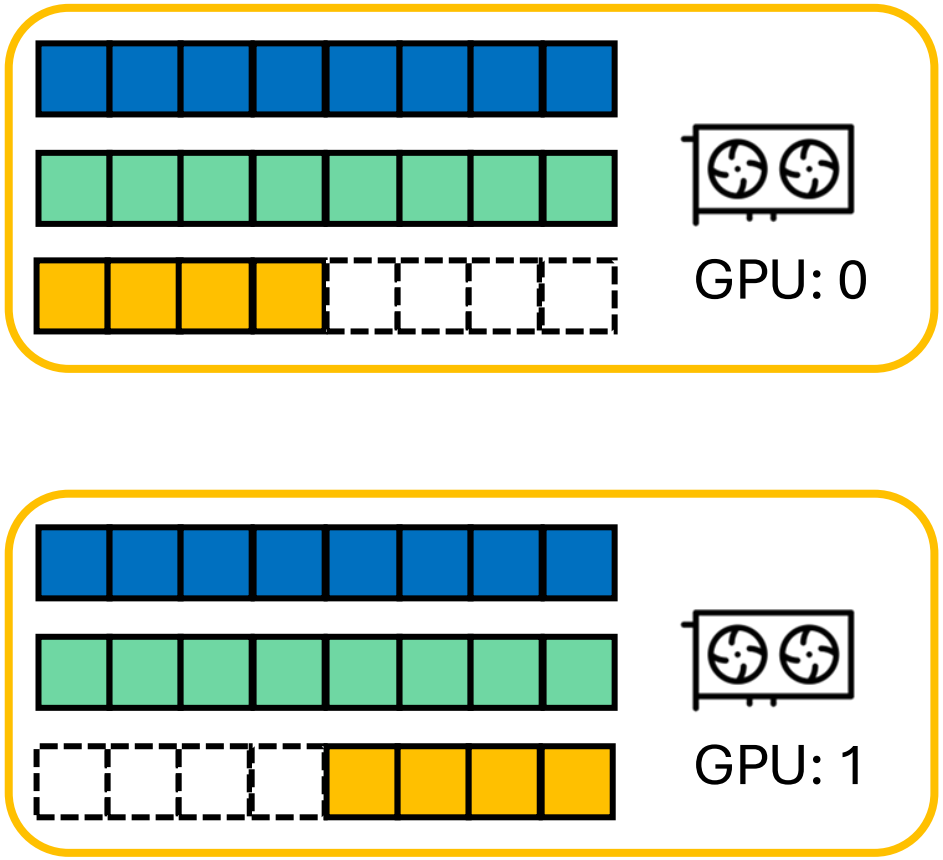
Reduce Gradients



Zero-3

Reduce Parameters

Zero-1: Reduce Optimizer



Reduce Optimizer States

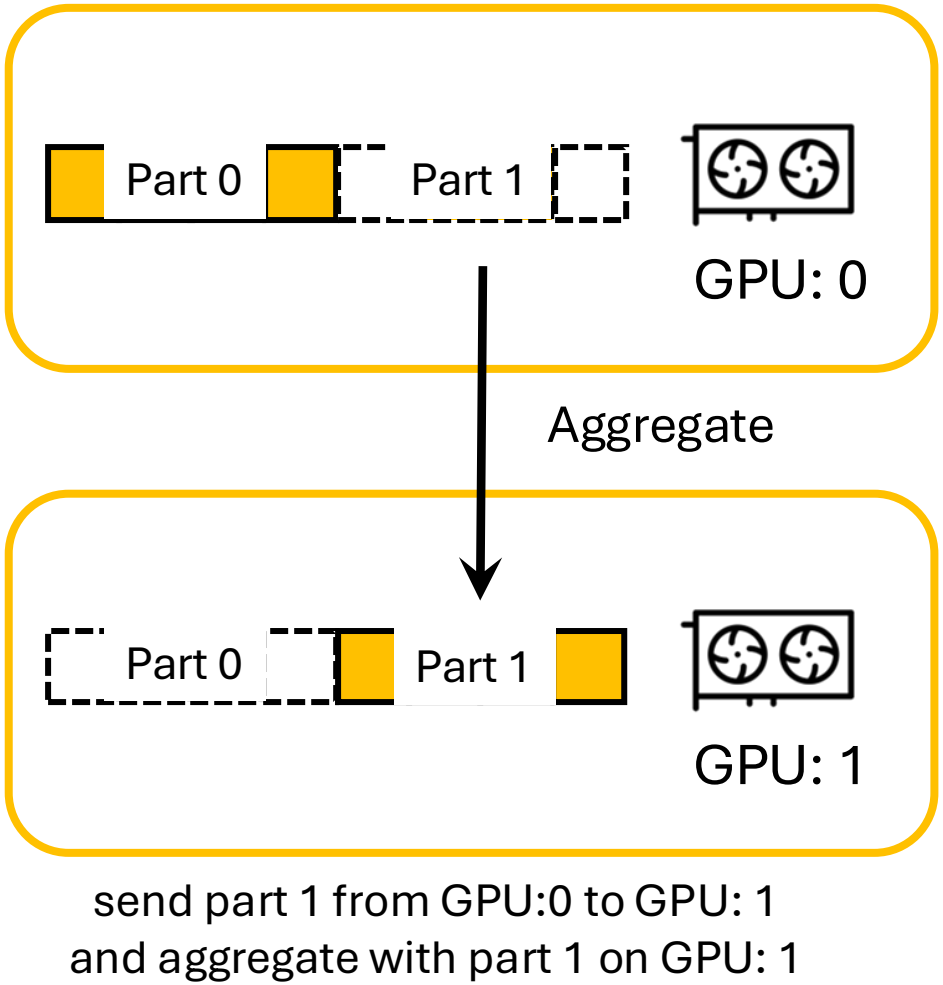
Each GPU keeps:

part of optimizer states,

whole copy of model parameters and gradients

Model Parameters
Gradients
Optimizer States

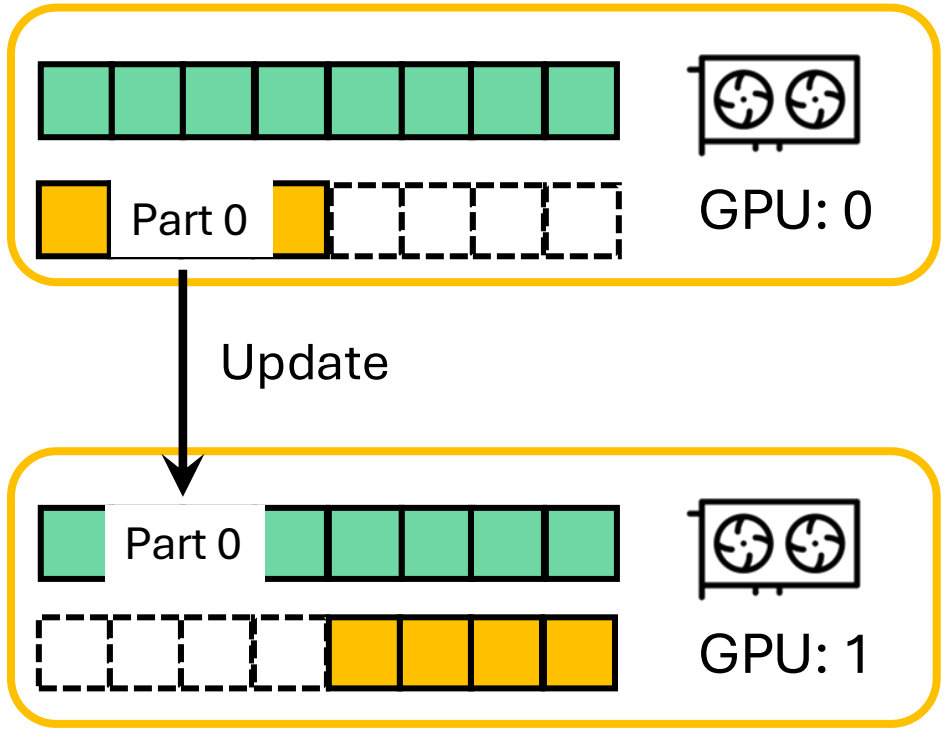
Zero-1: Reduce Optimizer



AllReduce:

Send the optimizer states to the corresponding GPU and aggregate

Zero-1: Reduce Optimizer

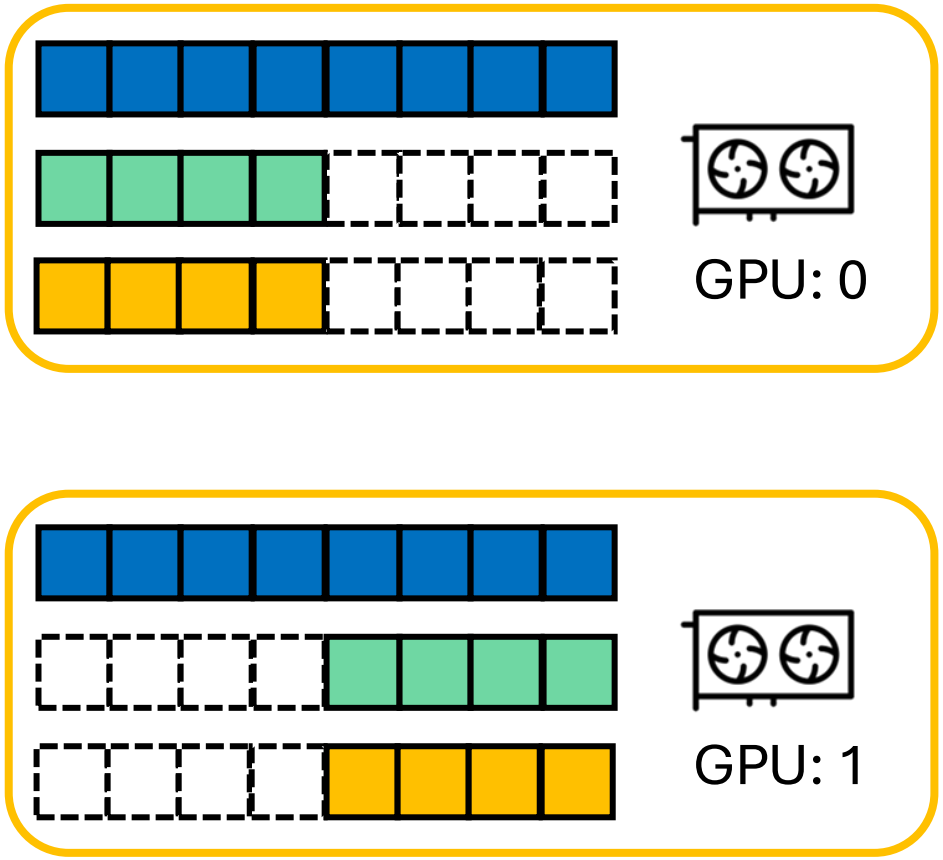


Update part 0 of gradients using part 0 of optimizer states

Gradient Update:

Update the gradient using corresponding optimizer states

Zero-2: Reduce Gradients



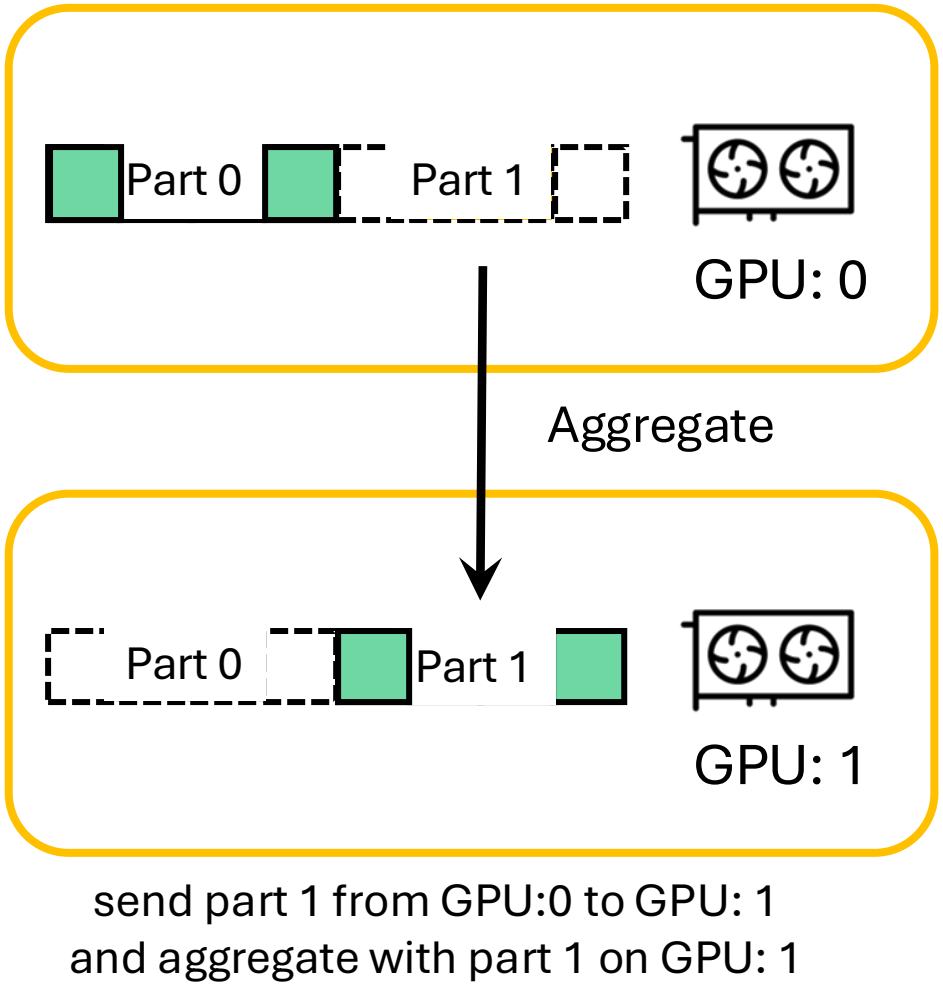
Reduce Optimizer States

Each GPU keeps:

part of optimizer states and gradients

whole copy of model parameters

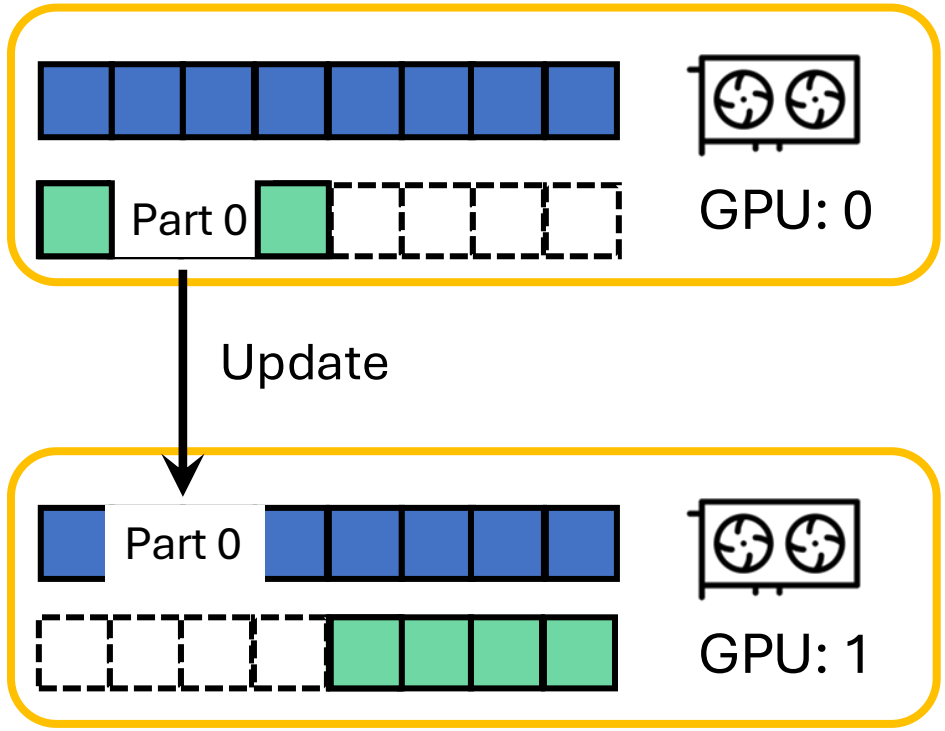
Zero-2: Reduce Gradients



AllReduce:

Send the gradients to the corresponding GPU and aggregate

Zero-2: Reduce Gradients

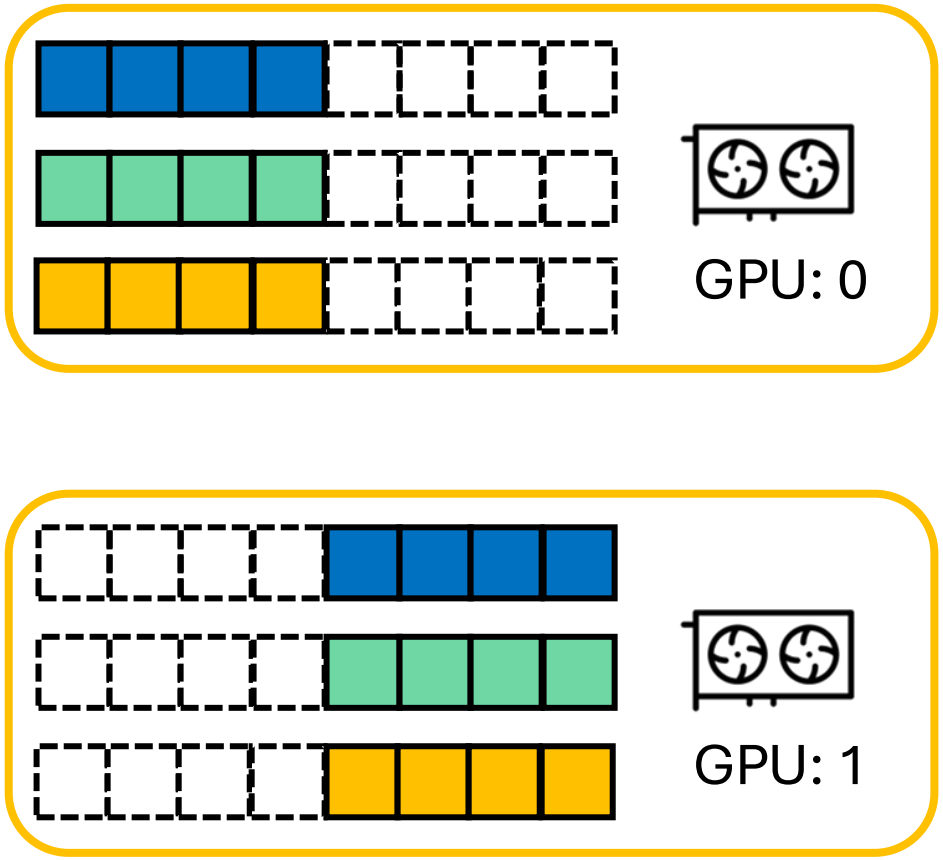


Parameter Update:

Update the parameters using corresponding gradients

* Gradient update same as Zero-1

Zero-3: Reduce Parameters

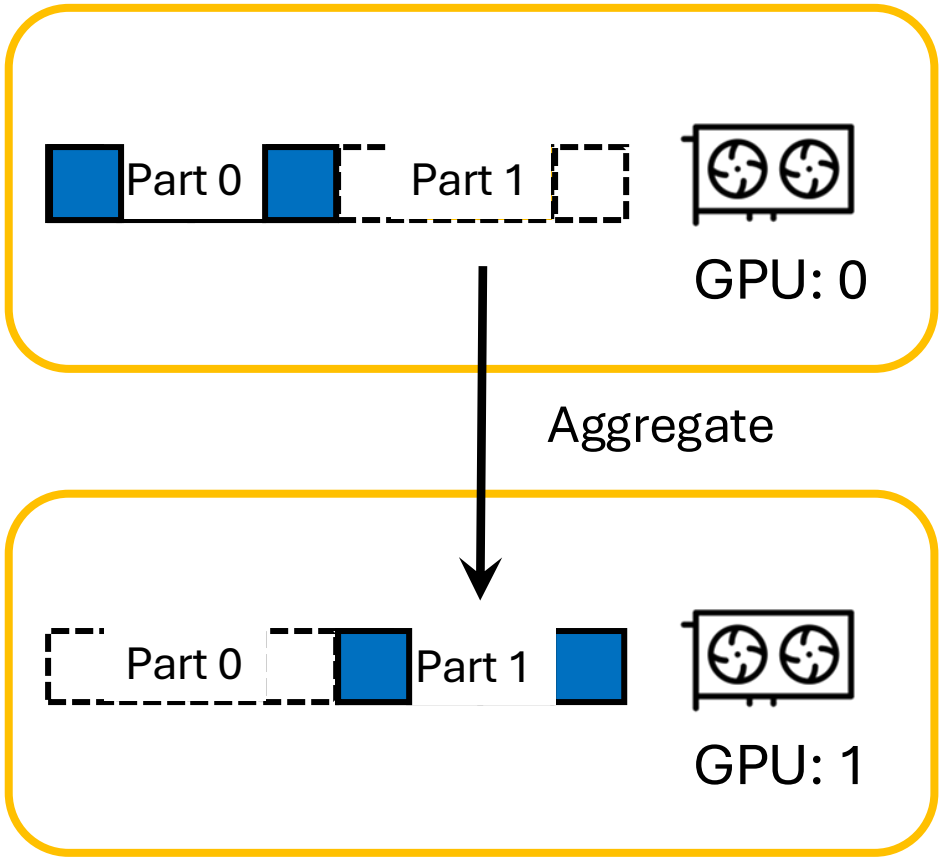


Reduce Optimizer States

Each GPU keeps:
part of optimizer states, gradients, and model parameters

Model Parameters
Gradients
Optimizer States

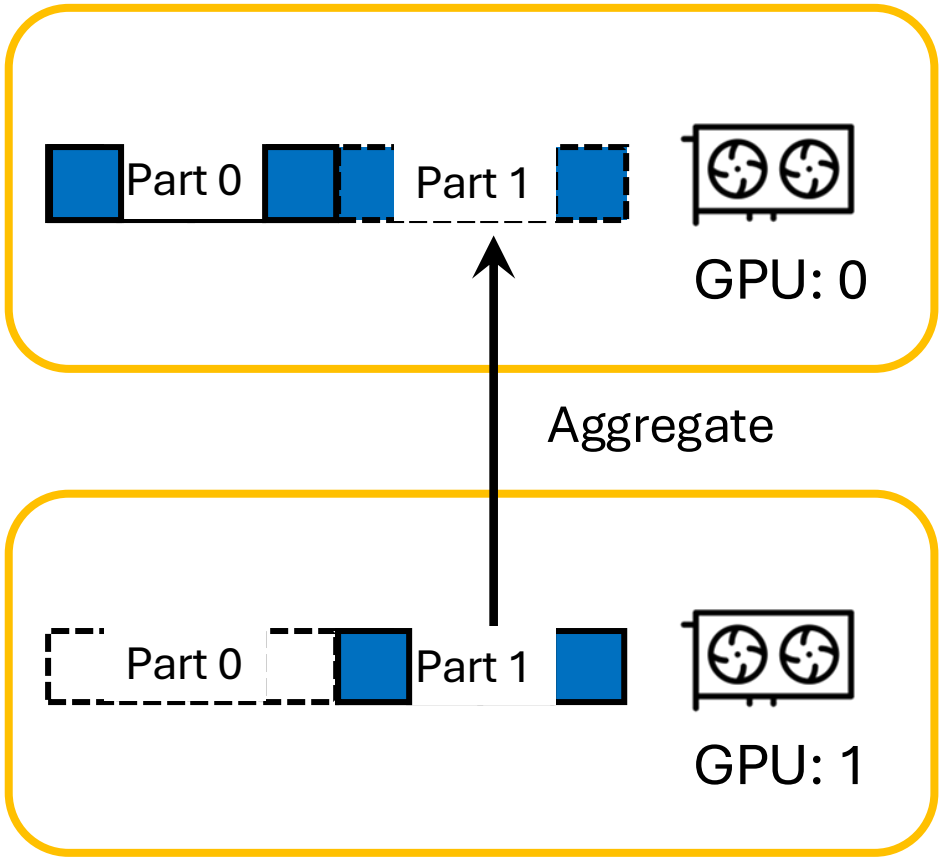
Zero-3: Reduce Parameters



AllReduce:

Send the **parameters** to the corresponding GPU and aggregate

Zero-3: Reduce Parameters



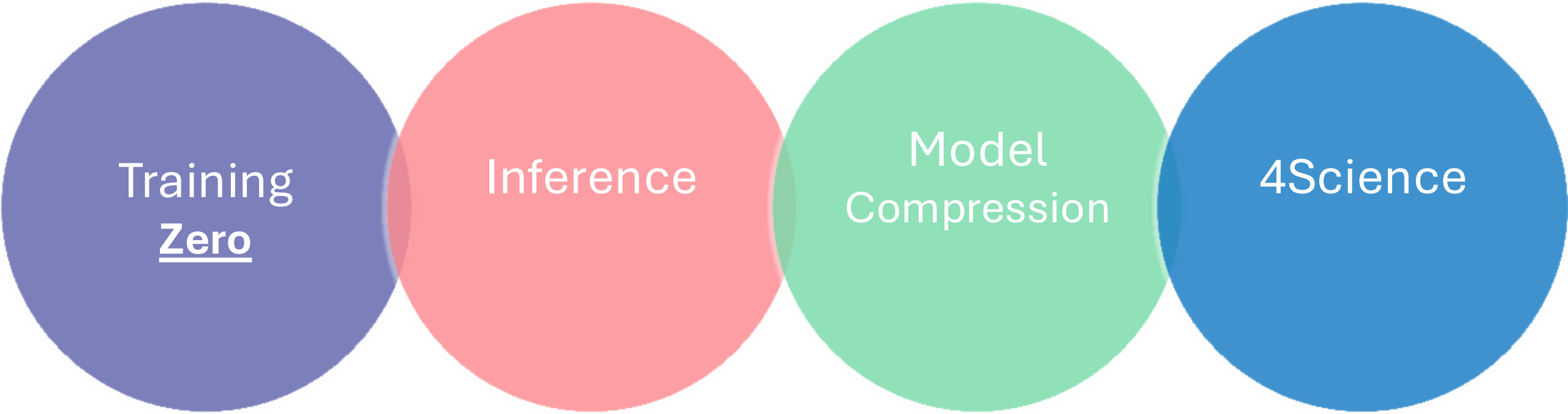
Forward and Backward:

Send the **parameters** to other GPU when needed

Part 6: DeepSpeed & Huggingface API

Intro to DeepSpeed

DeepSpeed: Optimization Library for DeepLearning



Define model and data as single GPU

```
# Import Library
import os
from transformers import Trainer, TrainingArguments

# Define the model and dataset
model = YourModel()
trn, val = TrainDataset(), ValidDataset()
```

```
# Define TrainingArguments
training_args = TrainingArguments(
  output_dir='./results', # store checkpoints
  num_train_epochs=YourEpochs,
  per_device_train_batch_size=YourBatchSize,
  per_device_eval_batch_size=YourBatchSize,
  # local_rank=int(os.environ['LOCAL_RANK']), # if DDP
  # deep_speed='./ds_config.json' # if Zero )
```

```
# Define Trainer
trainer = Trainer(
  model=model,
  args=TrainingArguments, # Previous Defined
  train_dataset=trn,
  eval_dataset=val)
```

Launch Training For Distributed Data Parallel

```
# Define Trainer
# Define node and port
head_node_ip = YourHeadNodeIP
port = YourPort
torchrun --nnodes=2 \ # number of nodes using
         --nproc_per_node=2 \ # number of GPU per node
         # …other configs
         YourTrainScript.py
```

Automatically find and allocate GPUs.

Launch Training For DeepSpeed

```
deepspeed YourTrainScript.py --deepspeed
          --deepspeed_config ds_config.json
```