Section 1 - Data Parallelism

1. Splitting Input Data

2023/05/25 (발표자: 정서형)

Table of Contents

- Overview
- Single-node training is too slow
- Data parallelism - the high-level bits
- Hyperparameter tuning

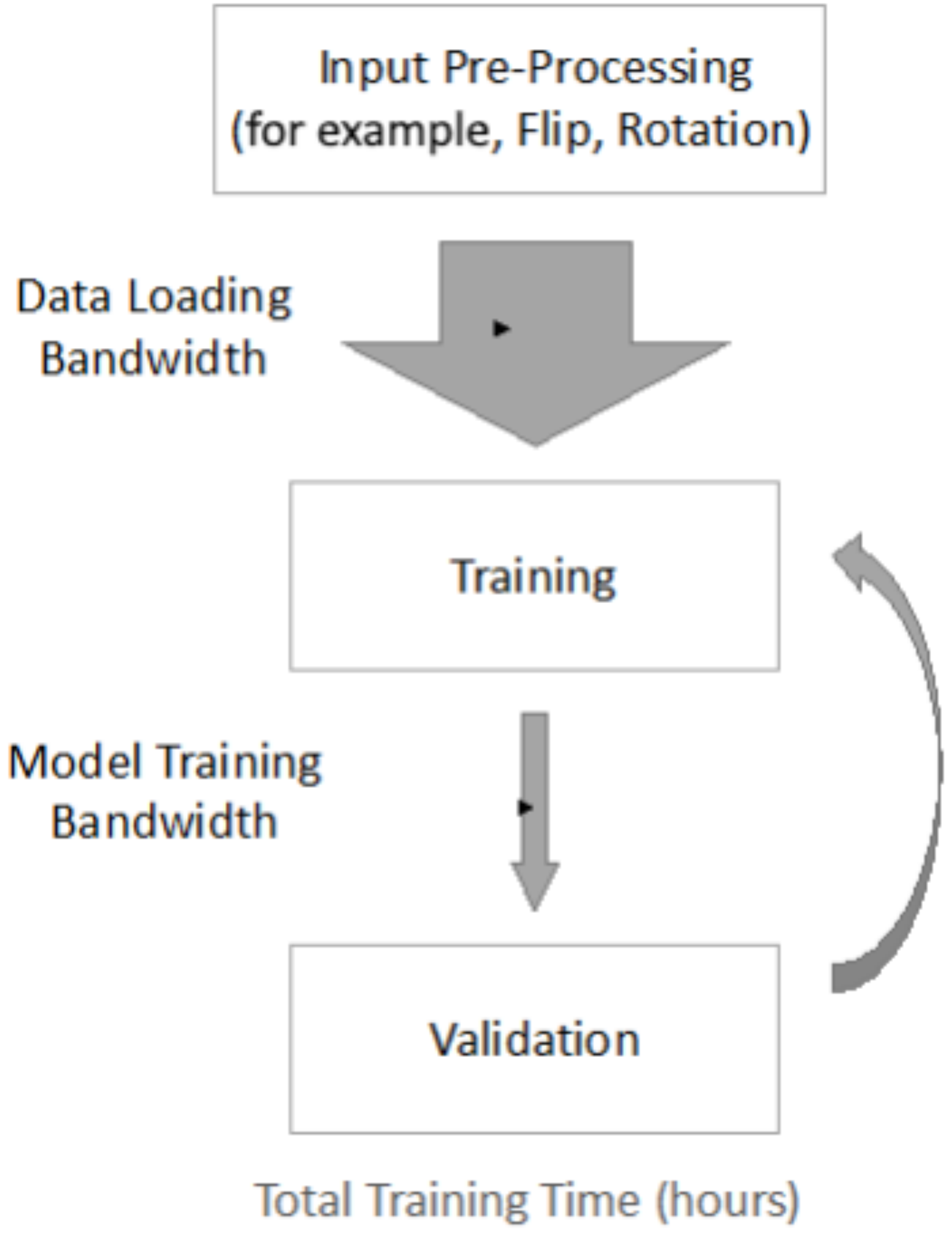
Overview

- Data Parallelism
  - Each GPU/node holds the full copy of a model
  - Partitions the input data into disjoint subsets
- Model synchronization
  - Since each GPU only trains its local model on a subset in data parallelism
  - Inference does not require model synchronization phase

Single-node training is too slow

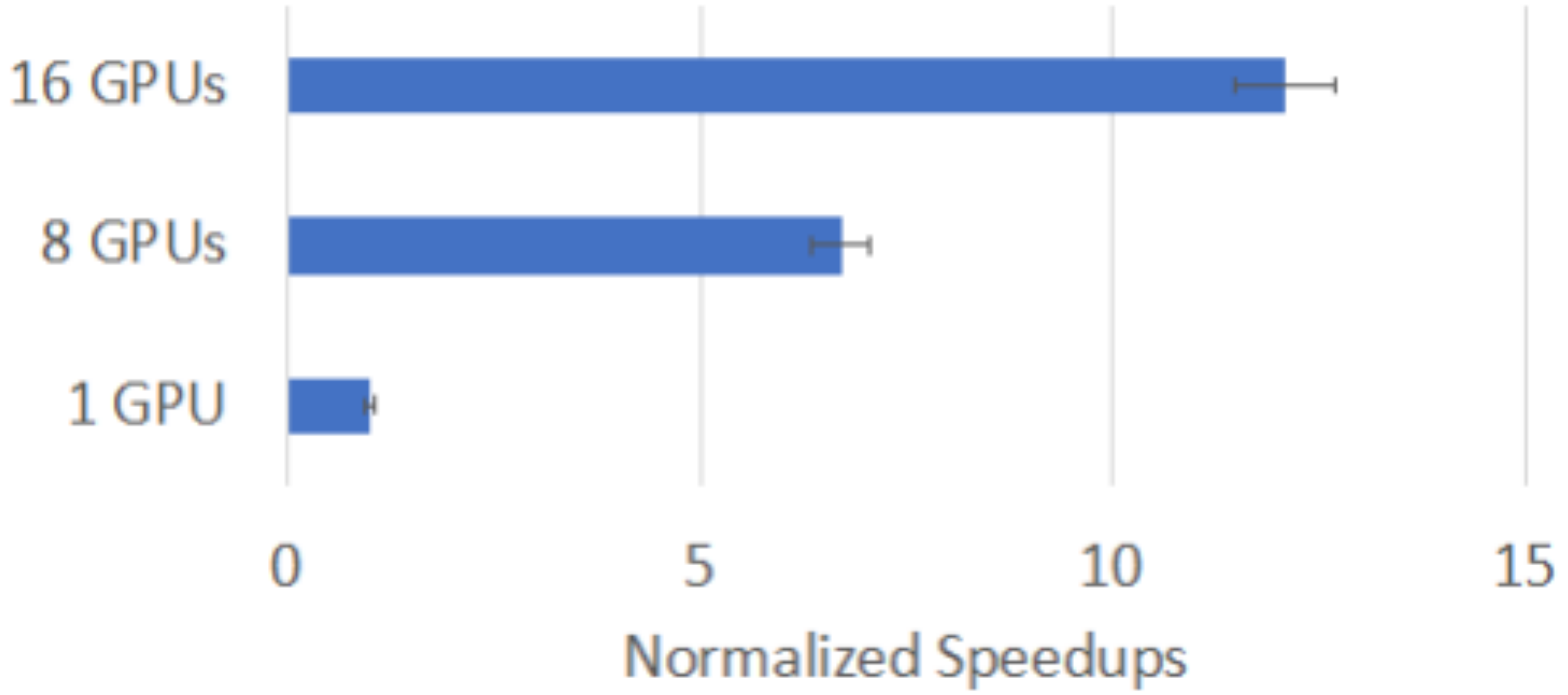
Bandwidth Mismatch

- Mismatch between data loading bandwidth and model training bandwidth makes single-node training too slow
- Due to the limited on-device memory of the GPUs or other accelerators



Single-node training is too slow

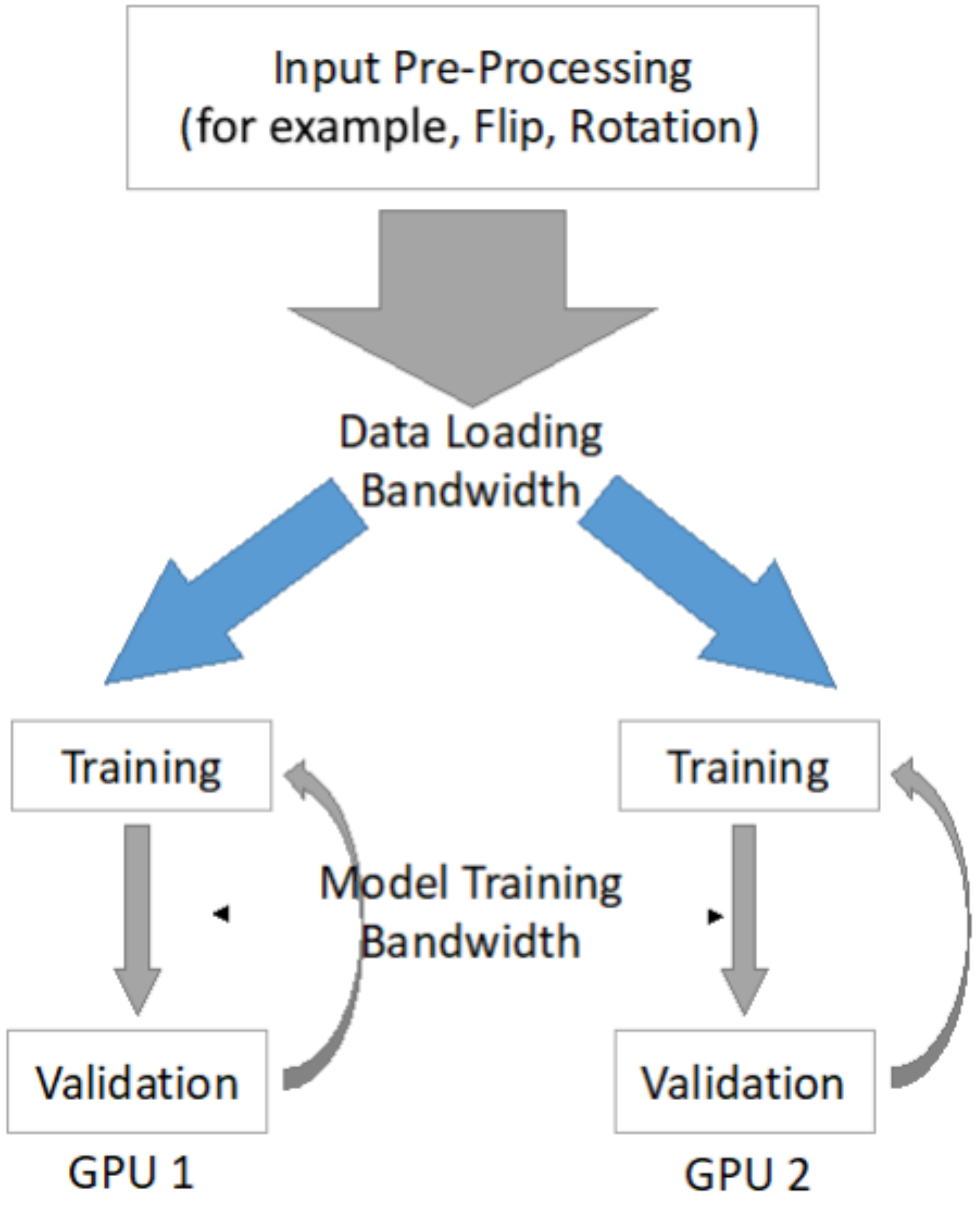
Accelerating the training with data parallelism



- By incorporating multiple GPUs into the same training job, we expand the model training bandwidth
- Ideally, the model training bandwidth should be linearly increased (system control overheads and network communications)

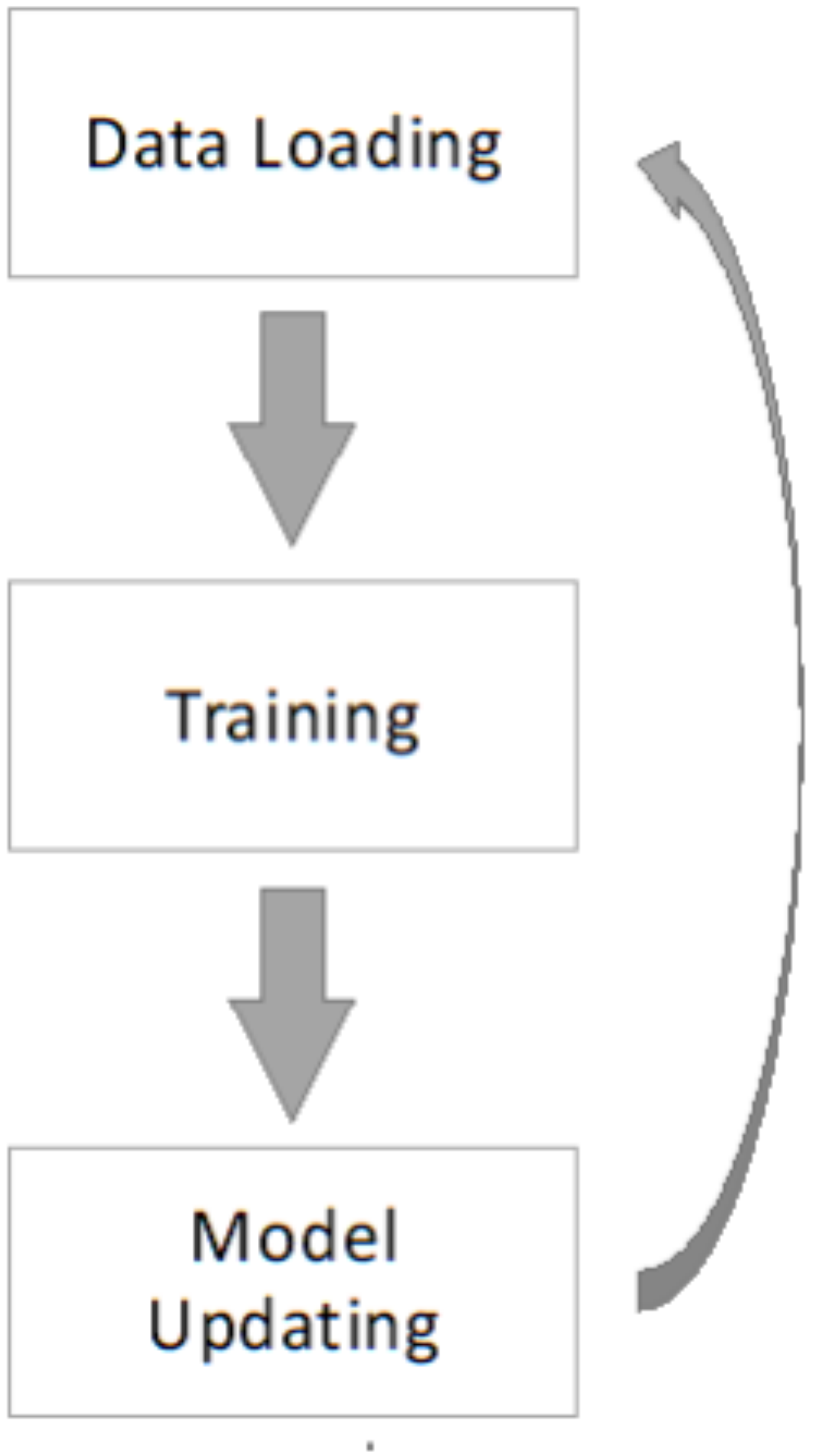
Data parallelism - the high-level bits

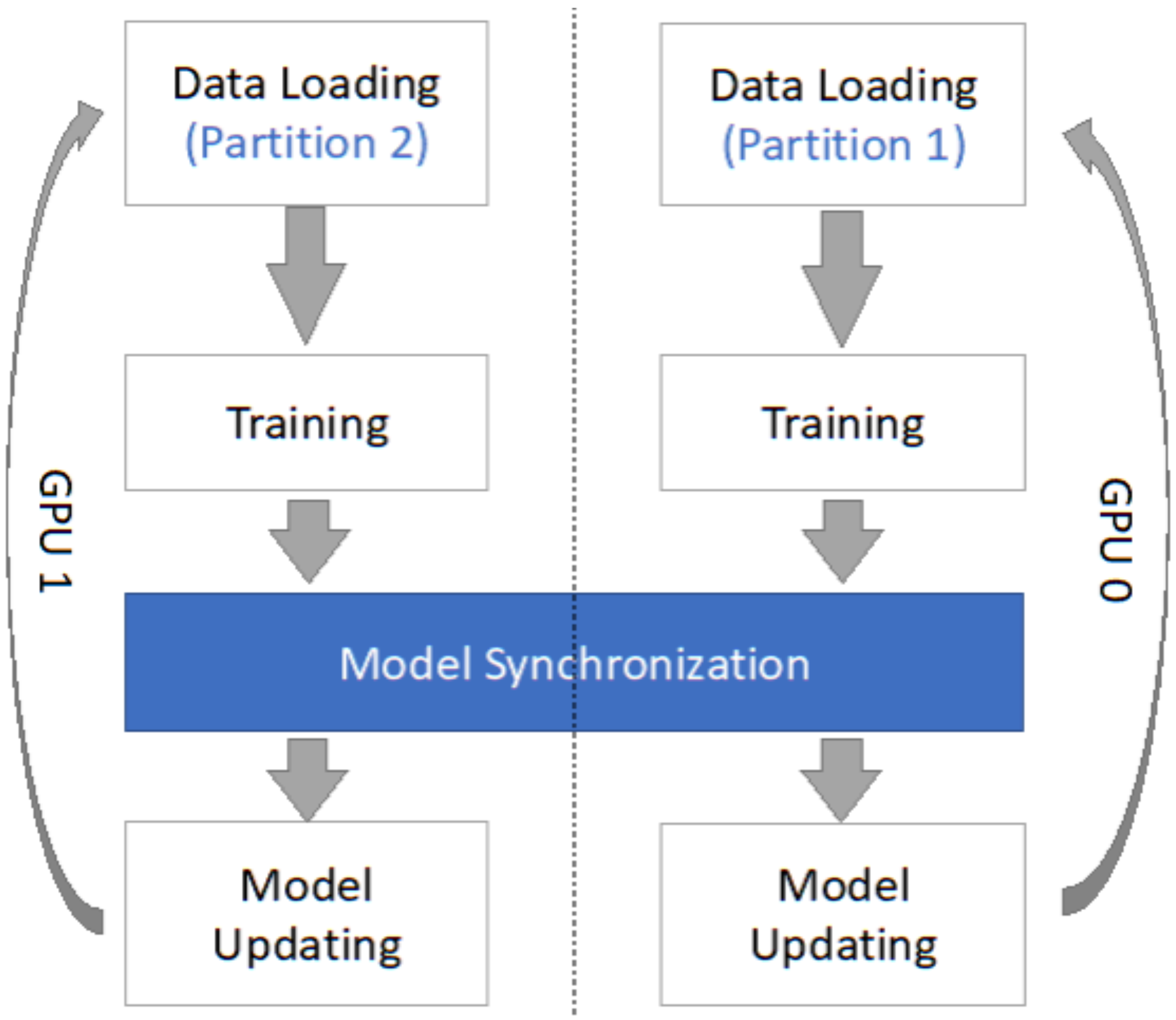
- Main difference between single-node training and data parallel training is how data loading bandwidth is spliced between multiple workers/GPUs



Data parallelism - the high-level bits

Single Node vs Data Parallelism





Data parallelism - the high-level bits

Stochastic Gradient Descent

- GD: compute the gradients over all the training data and update the model weights
- SGD: compute the gradients over a subset of all the training data and update the model weights
- In data parallelism, since each worker updates their model weights based on their local training data, the model parameters can be different after each of the training iterations
    - Need model synchronization

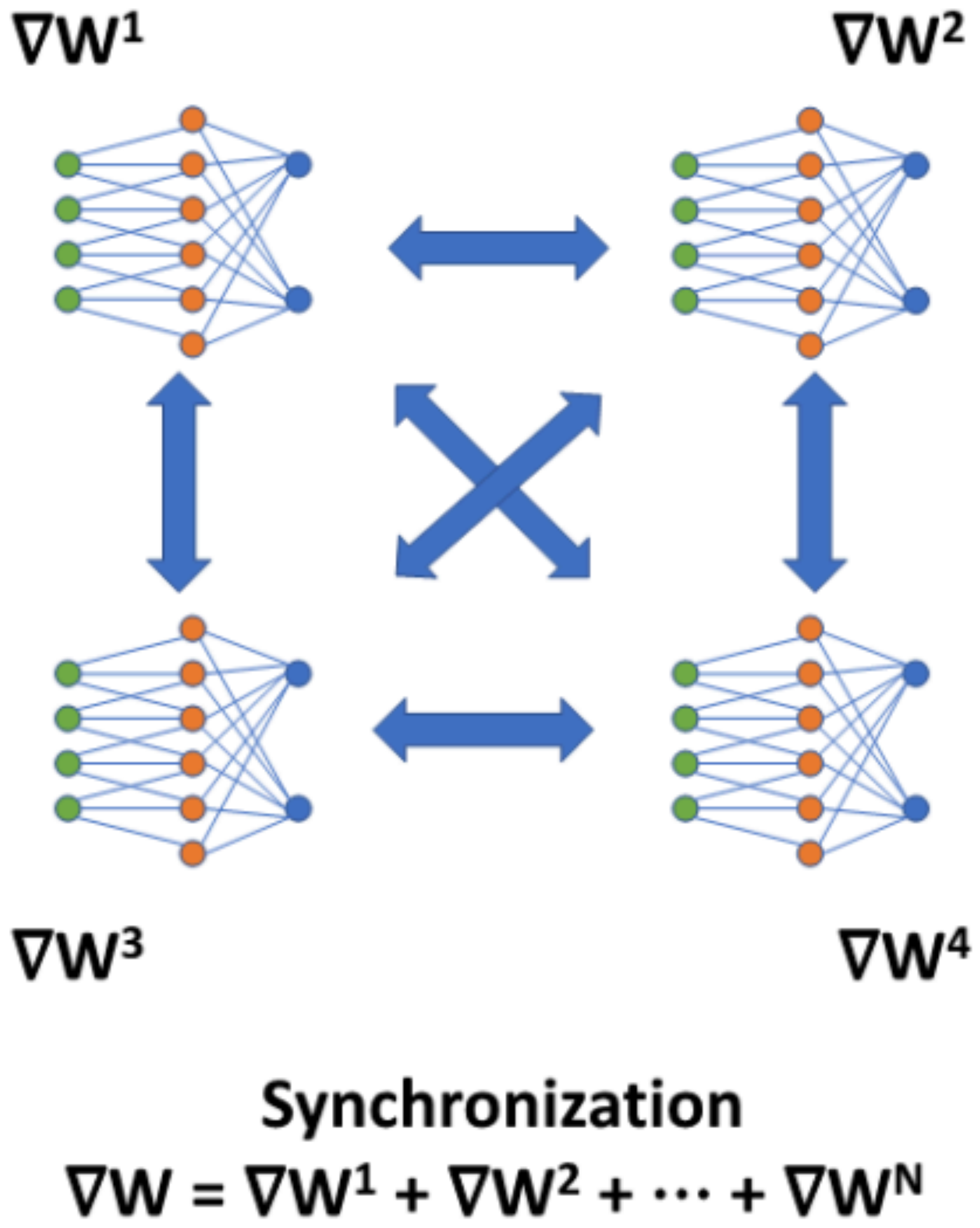
Data parallelism - the high-level bits

Model Synchronization

- Before the model parameter updates
  - Collect and sums up all the gradients from all the GPUs in use

$$\nabla W = \nabla W^1 + \nabla W^2 + \nabla W^3 + \ldots + \nabla W^N$$

  - Broadcast the aggregated gradients to all the GPUs
- For the real system implementations
  - Parameter Server Architecture
  - All-Reduce Architecture



Hyperparameter tuning

Global Batch Size

- Global Batch Size
  - In single node training: maximum that can fit into the accelerator's memory
  - In data parallel training: global batch size is not necessarily N*Max(single_node)
    - First hyper parameter to search
    - Too large: may not converge
    - Too small: waste of distributed computational resources

Hyperparameter tuning

Learning Rate Adjustment and Model Synchronization

- Rule of thumb for learning rate adjustment
  - Multiply the learning rate in the single-node case by the number of GPUs
- Model synchronization schemes (torch.distributed: https://pytorch.org/docs/stable/distributed.html)
  - NCCL
  - Glow
  - MPI

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✘	✓	?	✘	✓
recv	✓	✘	✓	?	✘	✓
broadcast	✓	✓	✓	?	✘	✓
all_reduce	✓	✓	✓	?	✘	✓
reduce	✓	✘	✓	?	✘	✓
all_gather	✓	✘	✓	?	✘	✓
gather	✓	✘	✓	?	✘	✓
scatter	✓	✘	✓	?	✘	✓
reduce_scatter	✘	✘	✘	✘	✘	✓
all_to_all	✘	✘	✓	?	✘	✓
barrier	✓	✘	✓	?	✘	✓