MIT HAN LAB

Lecture 13
Distributed Training

Song Han
songhan@mit.edu

Lecture Plan

Understand how distributed training works and improve the efficiency

Today we will:

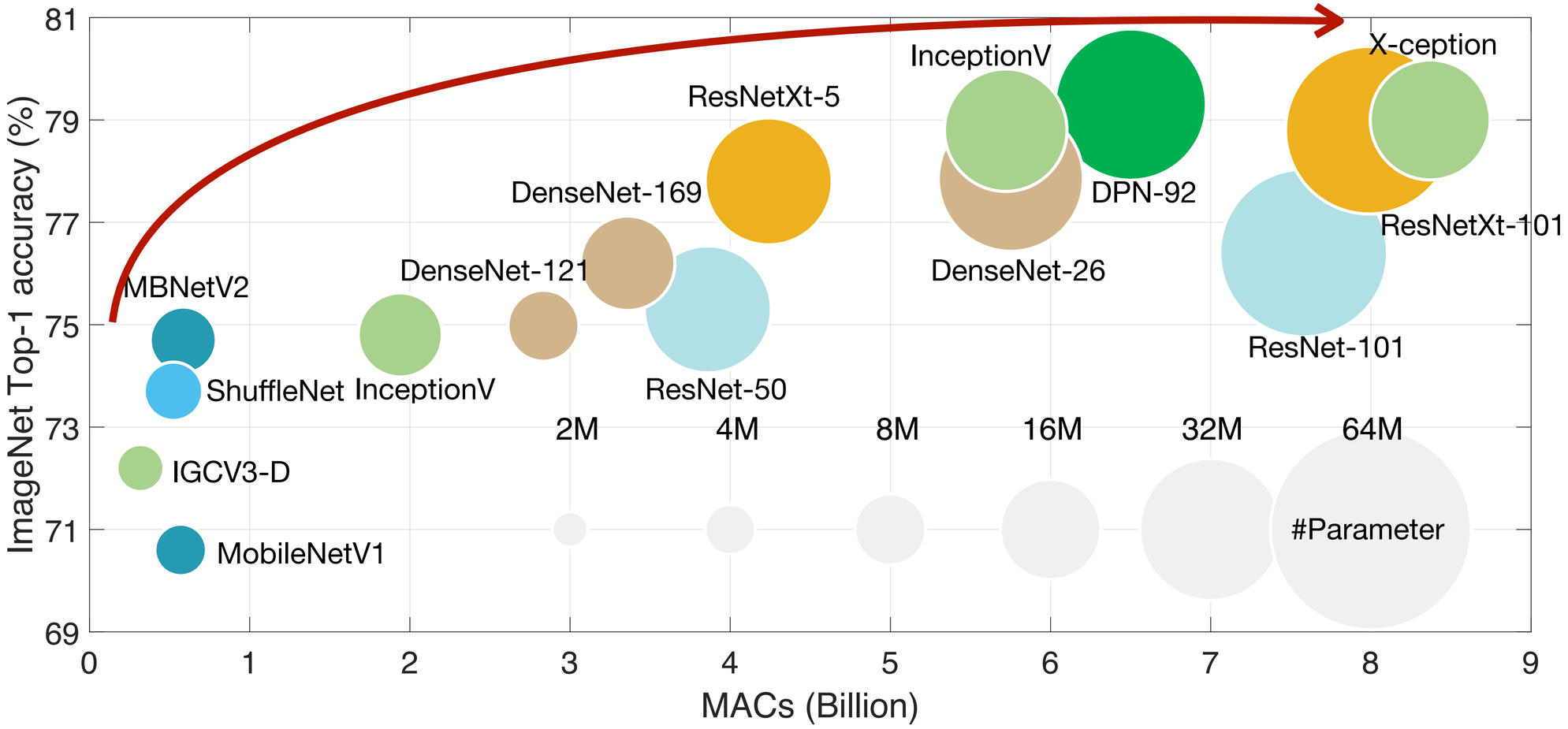
1. Background and motivation of distributed training
2. Introduce the basic concepts of distributed training
3. Distributed training with data parallelism
4. Distributed communication schemes
5. Distributed training with model parallelism

Section 1: Background

What is distributed training and why we need it?

Models are getting larger and larger

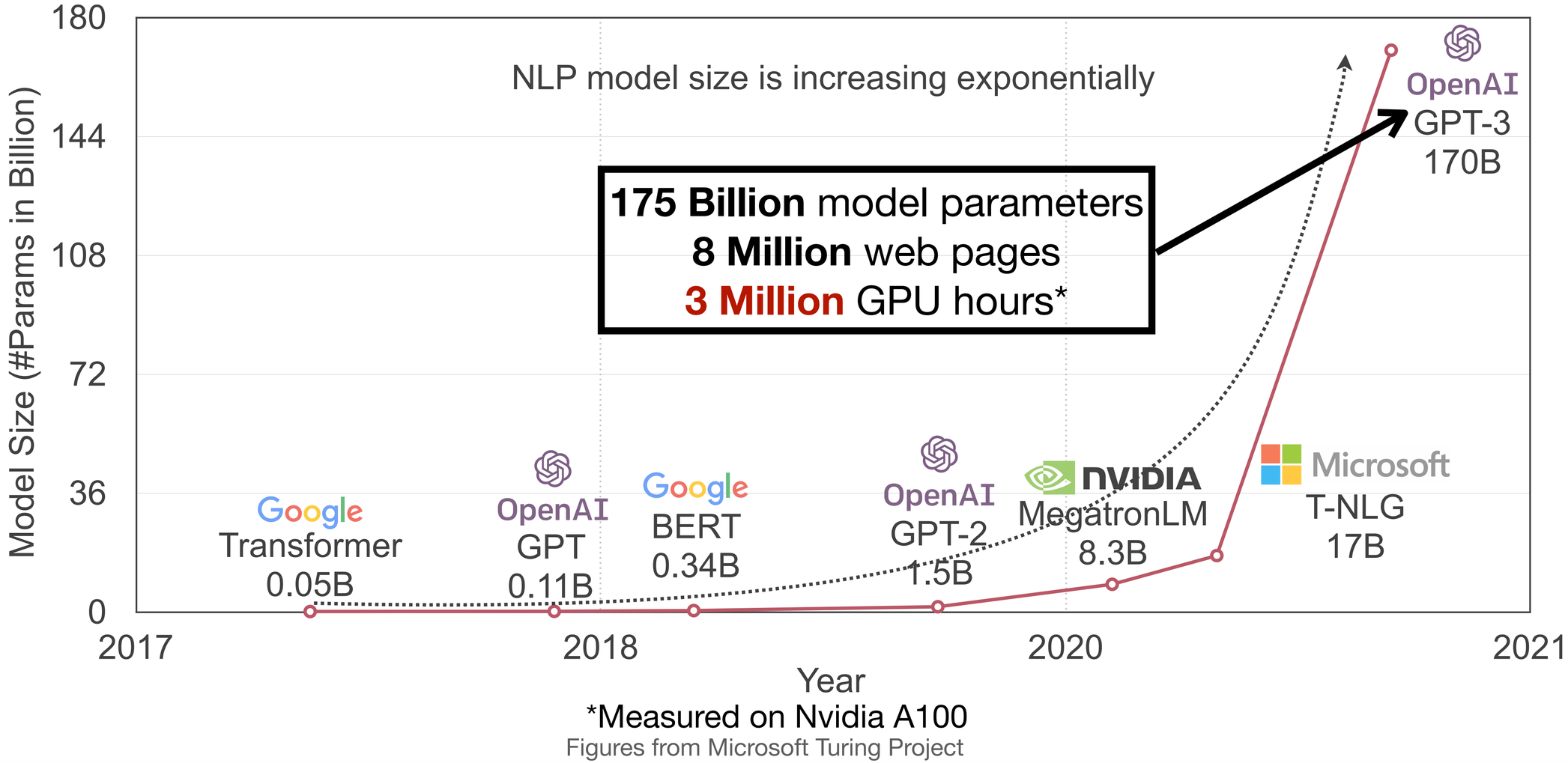
Better model always comes with higher computational cost (vision)



Figures from Once-for-all project page.

Models are getting larger and larger

Better model always comes with higher computational cost (NLP)



Figures from Microsoft Turing Project

Models are getting larger and larger

Large Models Take Longer Time to Train

Models	#Params (M)	Training Time (GPU Hours)
ResNet-50	26	31
ResNet-101	45	44
BERT-Base	108	84
Turing-NLG 17B	17,000	TBA
GPT-3 175B	175,000	3,100,000

Measured on Nvidia A100

If without distributed training, a single GPU would take **335 years** to finish GPT-3!

Models are getting larger and larger

Large Models Take Longer Time to Train

Boss: What did you do last month?

You: Trained the model for one epoch. 

Boss: Umm, fine, what is your plan for next month?

You: Train… train the model for one more epoch? 

Distributed Training is Necessary

- Developers / Researchers' time **are more valuable** than hardware .
- If a training takes **10 GPU days**
  - Parallelize with distributed training
  - 1024 GPUs can finish in 14 minutes (ideally)!
- The develop and research cycle will be greatly boosted

Let's see a use case of distributed training!

Large-Scale Distributed Training with Super Computer

SUMMIT Super Computer:

- CPU: 2 x 16 Core IBM POWER9 (connected via dual NVLINK bricks, 25GB/s each side)
- GPU: 6 x NVIDIA Tesla V100
- RAM: 512 GB DDR4 memory
- Data Storage: HDD
- Connection: Dual-rail EDR InfiniBand network of 23 GB/s

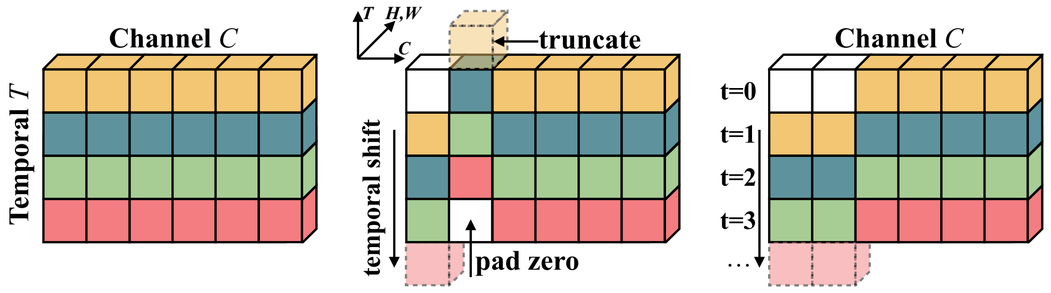
Large-Scale Distributed Training with Super Computer

Train a Vision Model on 660 Hours of Videos

Models	Training Time	Accuracy
1 SUMMIT Nodes (6 GPUs)	49h 50min	74.1%
128 SUMMIT Nodes (768 GPUs)	28min	74.1%
256 SUMMIT Nodes (1536 GPUs)	14min (**211x**)	74.0%

Speedup the training by 200x, **from 2 days to 14minutes.**

- **Model setup**: 8-frame ResNet-50 TSM for video recognition
- **Dataset**: Kinetics (240k training videos) x 100 epoch



TSM: Temporal Shift Module for Efficient Video Understanding [Lin 2019]

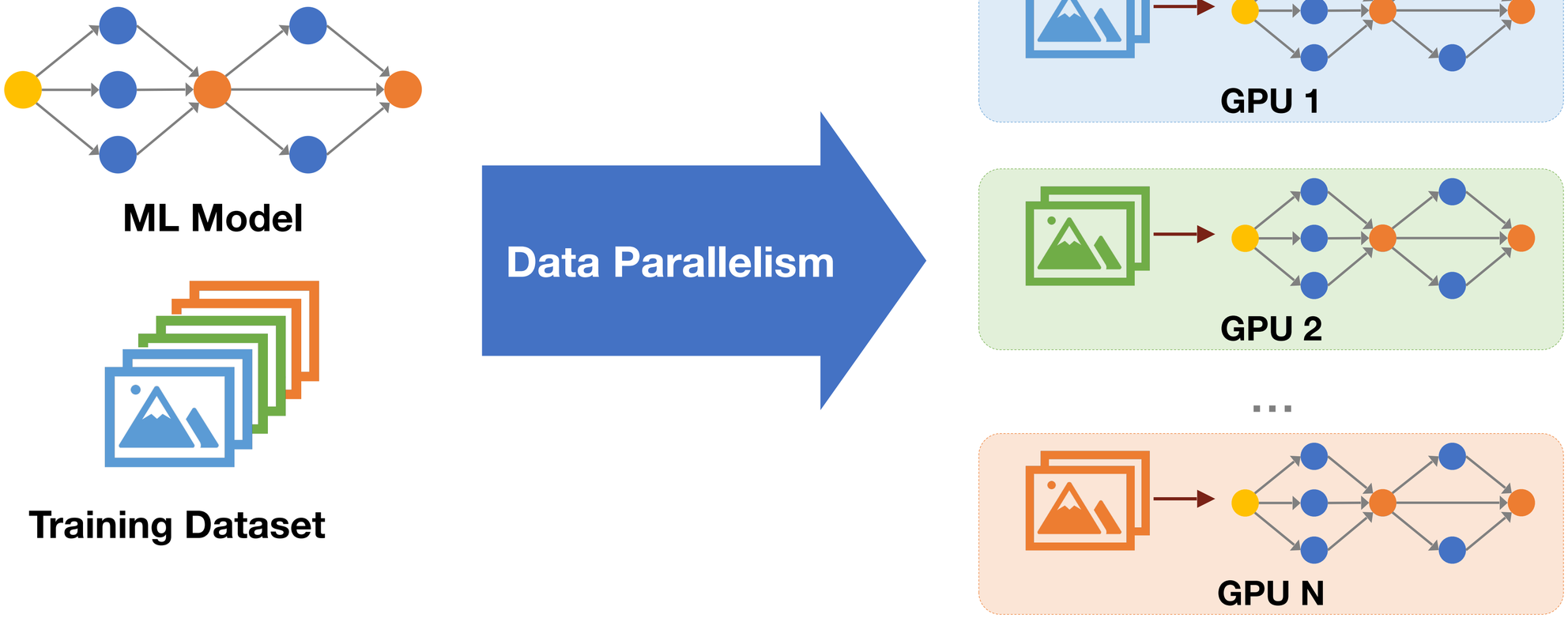
Section 2: Distributed Training Basics

Parallelism in Distributed Training

- **Data Parallelism**
- Model Parallelism
- Compare the Advantages and Disadvantages of Two Parallelism

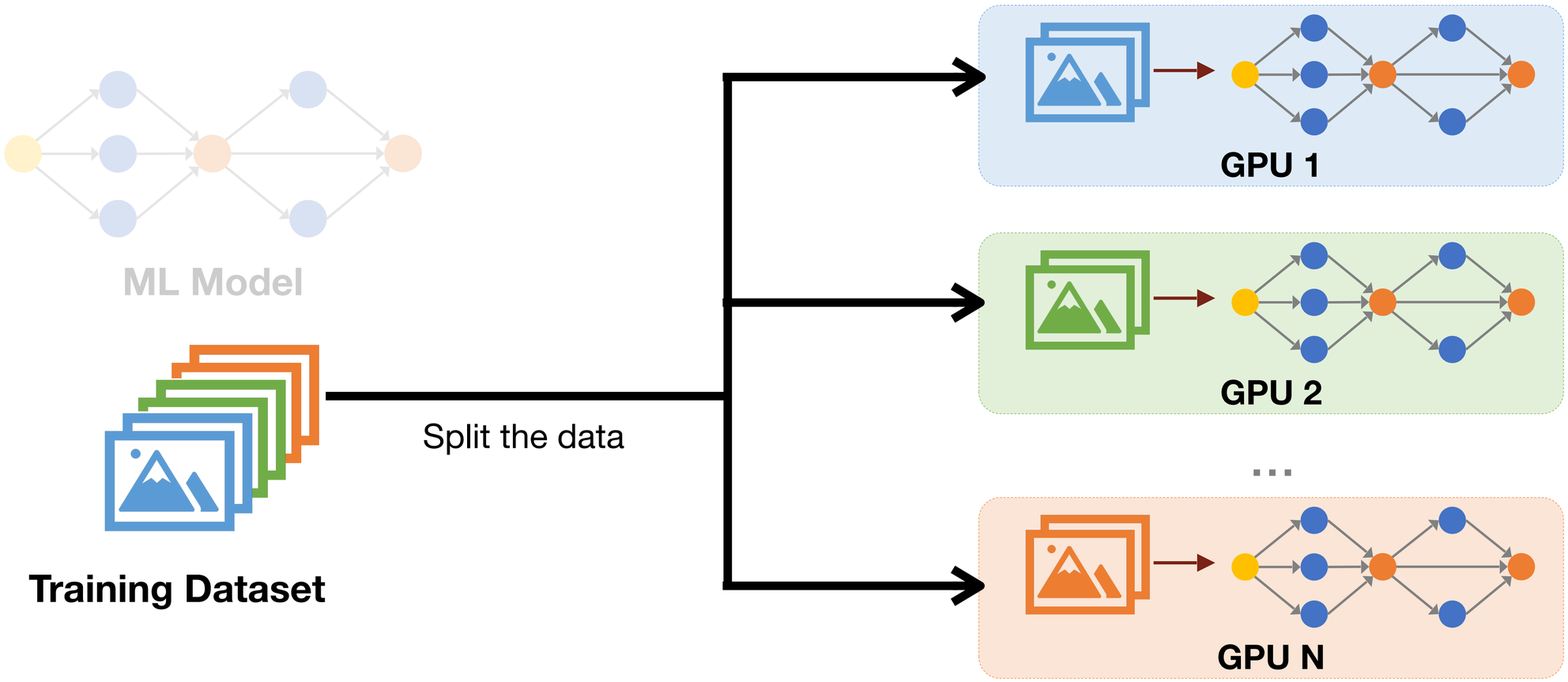
Introduction to Distributed Training

Data Parallelism



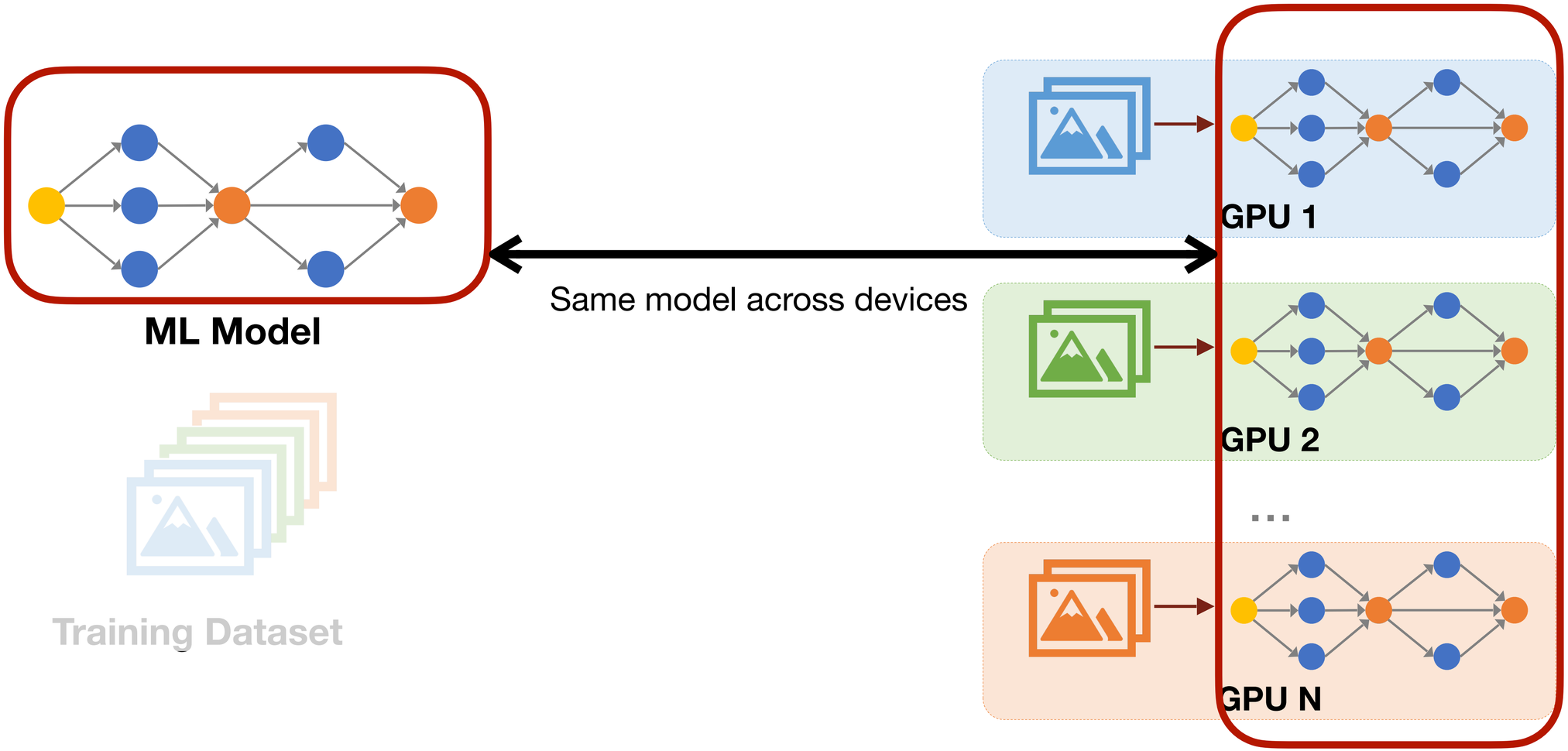
Introduction to Distributed Training

Data Parallelism



Introduction to Distributed Training

Data Parallelism



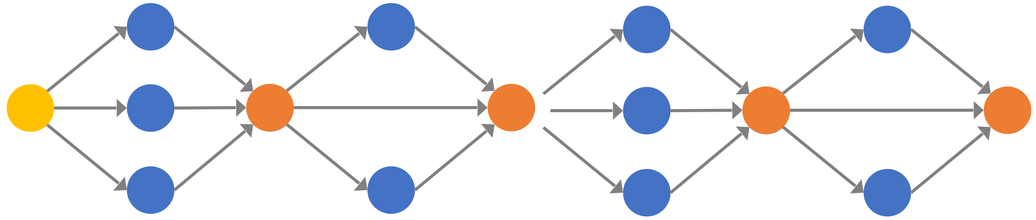
Parallelism in Distributed Training

- Data Parallelism
- **Model Parallelism**
- Compare the Advantages and Disadvantages of Two Parallelism

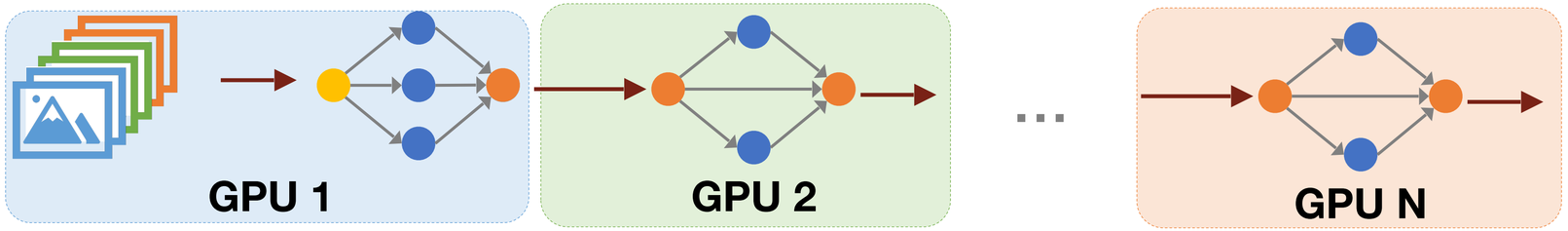
Introduction to Distributed Training

Model Parallelism

Training Dataset



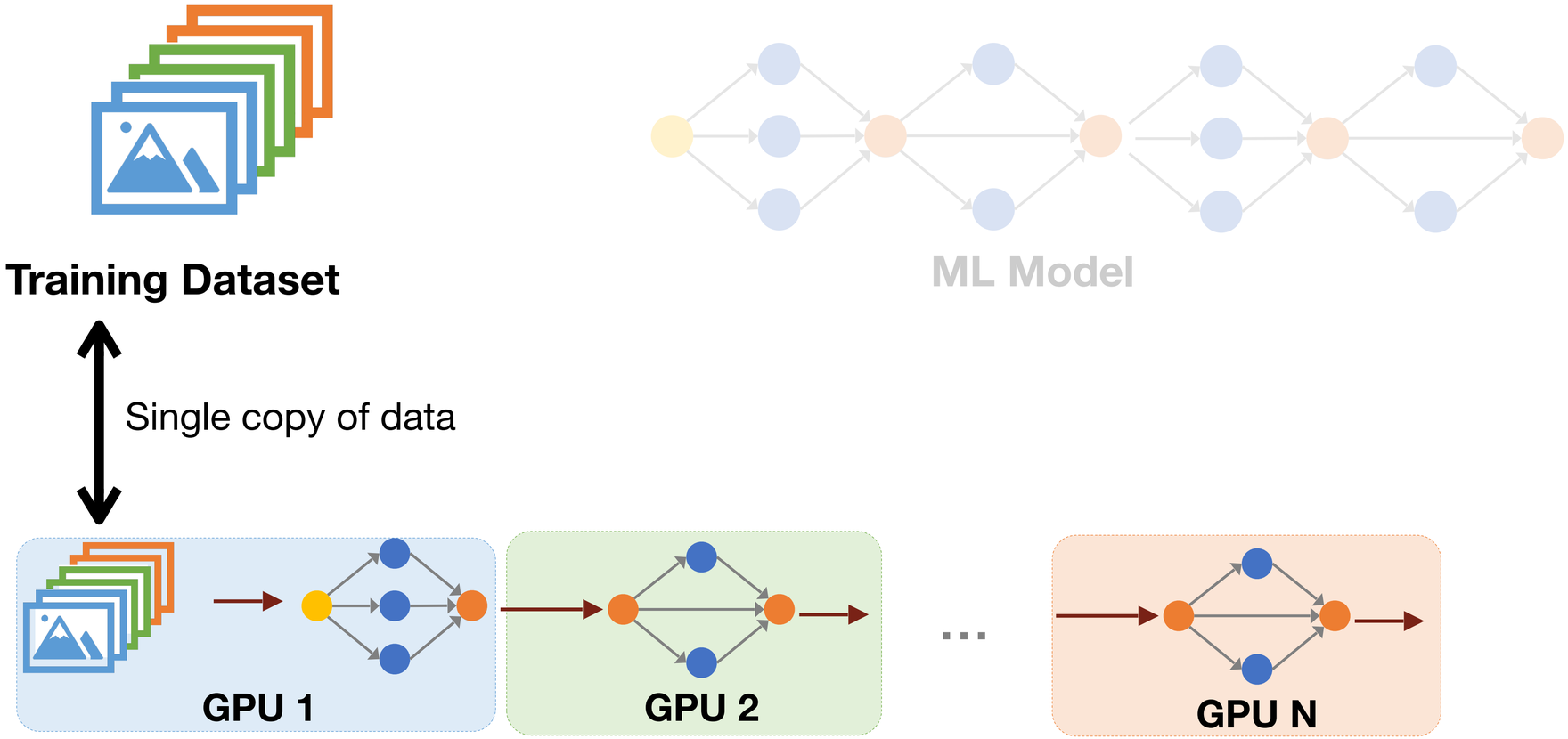
ML Model



Figures credit from CMU 15-849 [Jia 2022]

Introduction to Distributed Training

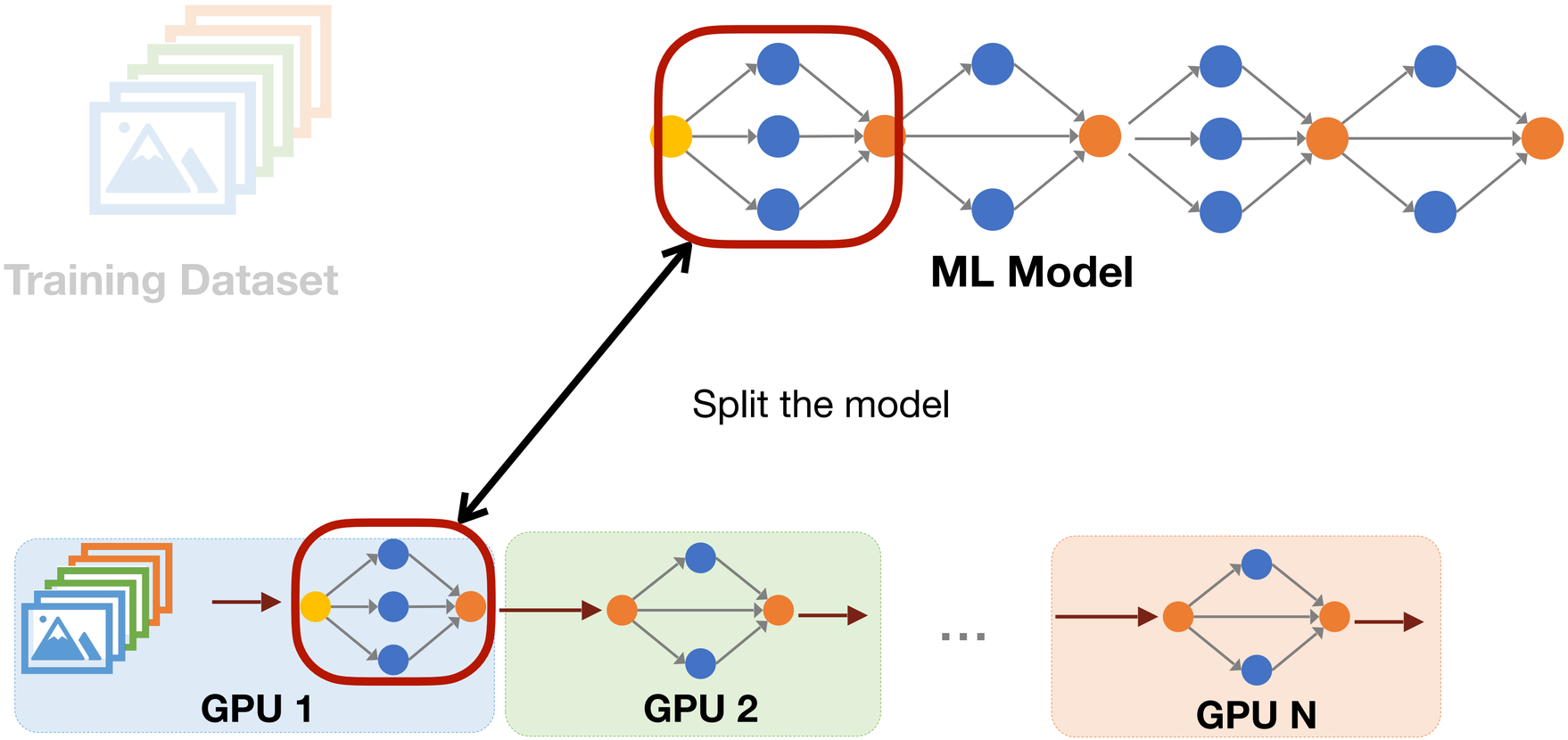
Model Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Introduction to Distributed Training

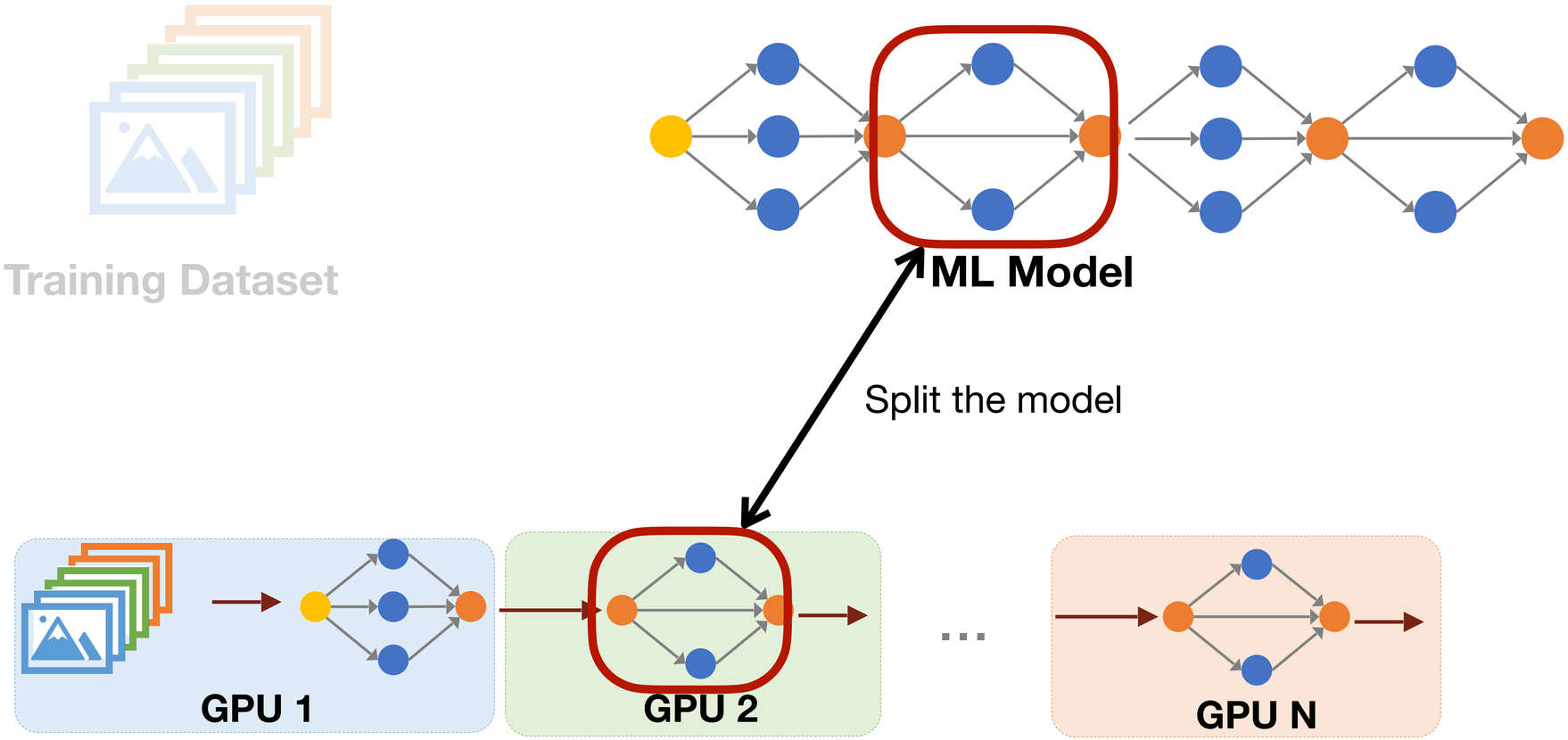
Model Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Introduction to Distributed Training

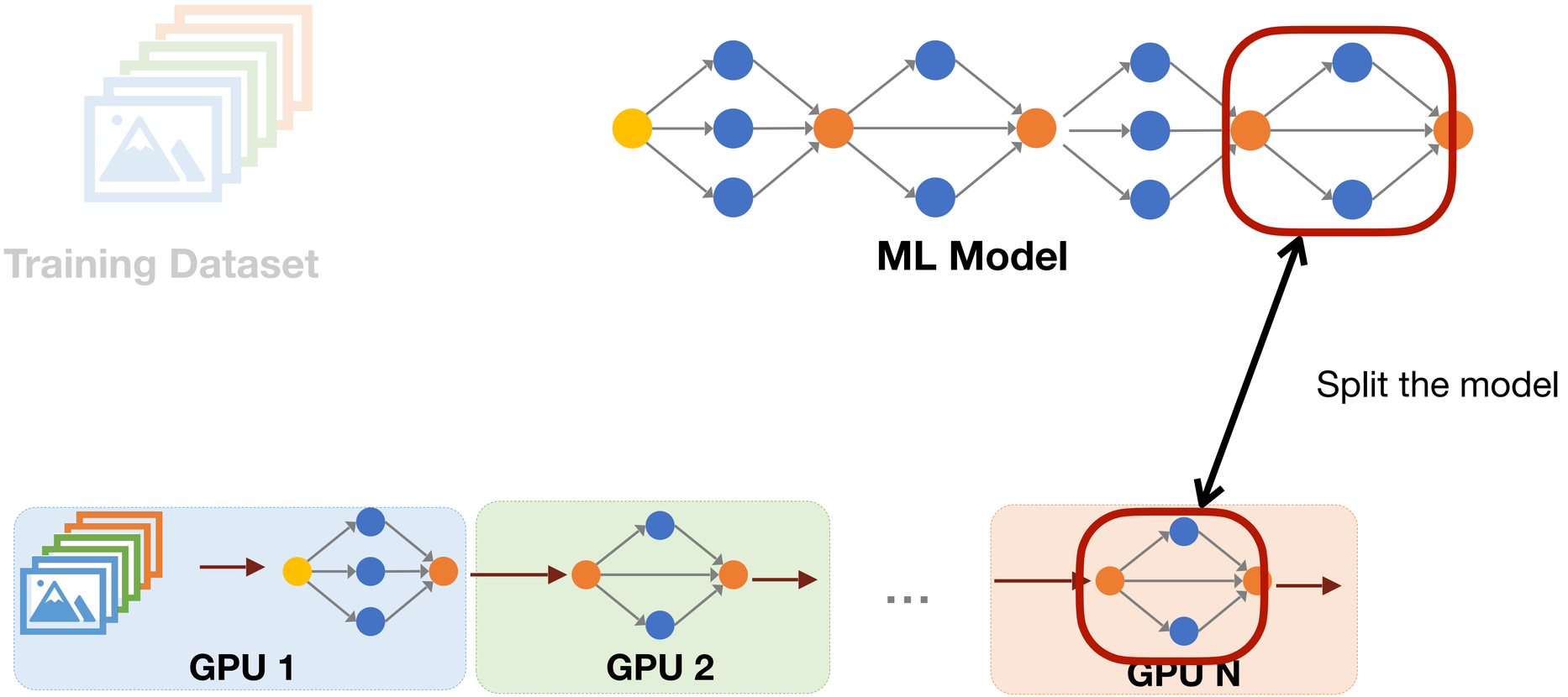
Model Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Introduction to Distributed Training

Model Parallelism

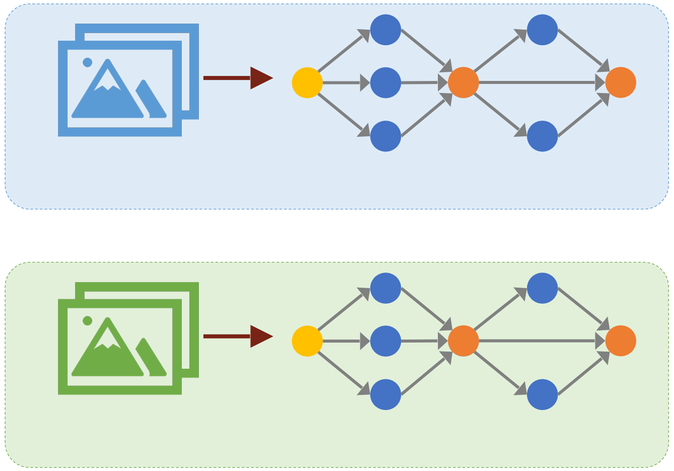


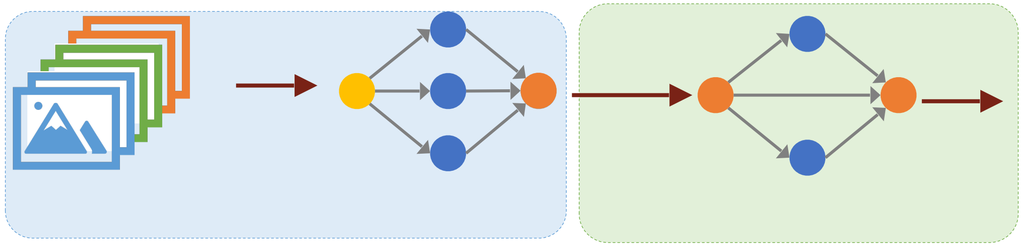
Figures credit from CMU 15-849 [Jia 2022]

Parallelism in Distributed Training

- Data Parallelism
- Model Parallelism
- **Compare the Advantages and Disadvantages of Two Parallelism**

Comparison between two parallelism





Data Parallelism:

- Split the data
- Same model across devices
- Easy to parallelize, high utilization
- N copies of model

Model Parallelism:

- Split the model
- Move activations through devices
- Hard to parallelize, load balancing issue
- Single copy of model

Figures credit from CMU 15-849 [Jia 2022]

Section 3: Distributed Training with Data Parallelism

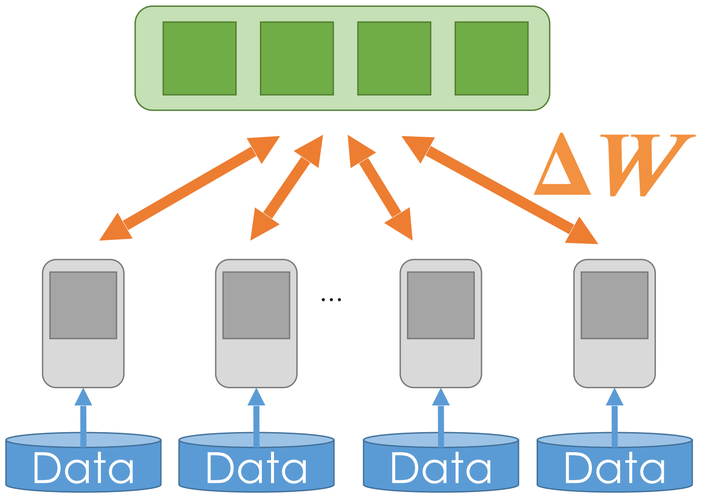
Understanding distributed training workflow and system design challenges.

Dive into Data Parallelism

Scaling Distributed Machine Learning with the Parameter Server

Parameter Server

The central controller of the whole training process



Worker nodes

The hardware accelerators and dataset storage.

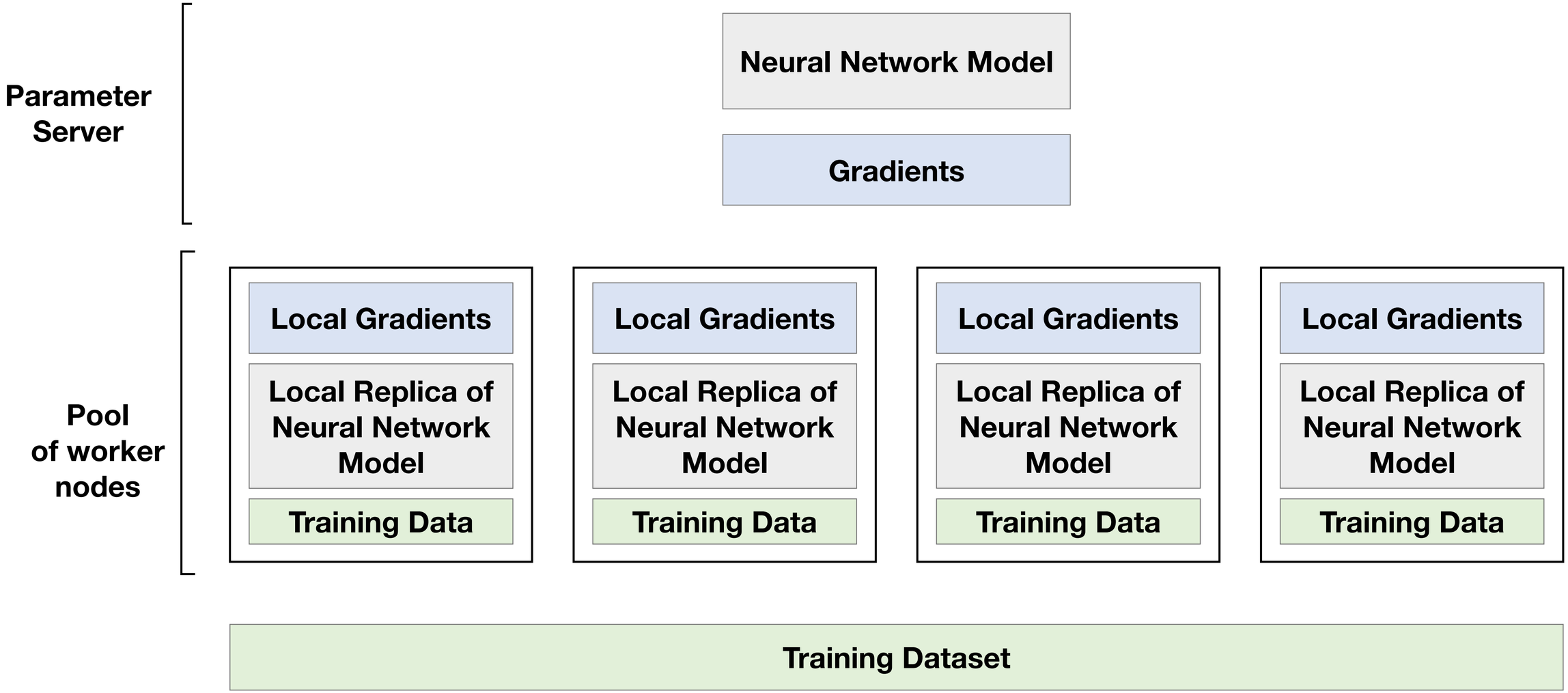
Two different roles in framework:

- Parameter Server: receive gradients from workers and send back the aggregated results
- Workers: compute gradients using splitted dataset and send to parameter server

Scaling Distributed Machine Learning with the Parameter Server. Mu Li et al. 2014
Figure credits from: Deep Gradient Compression. Lin et al. 2018

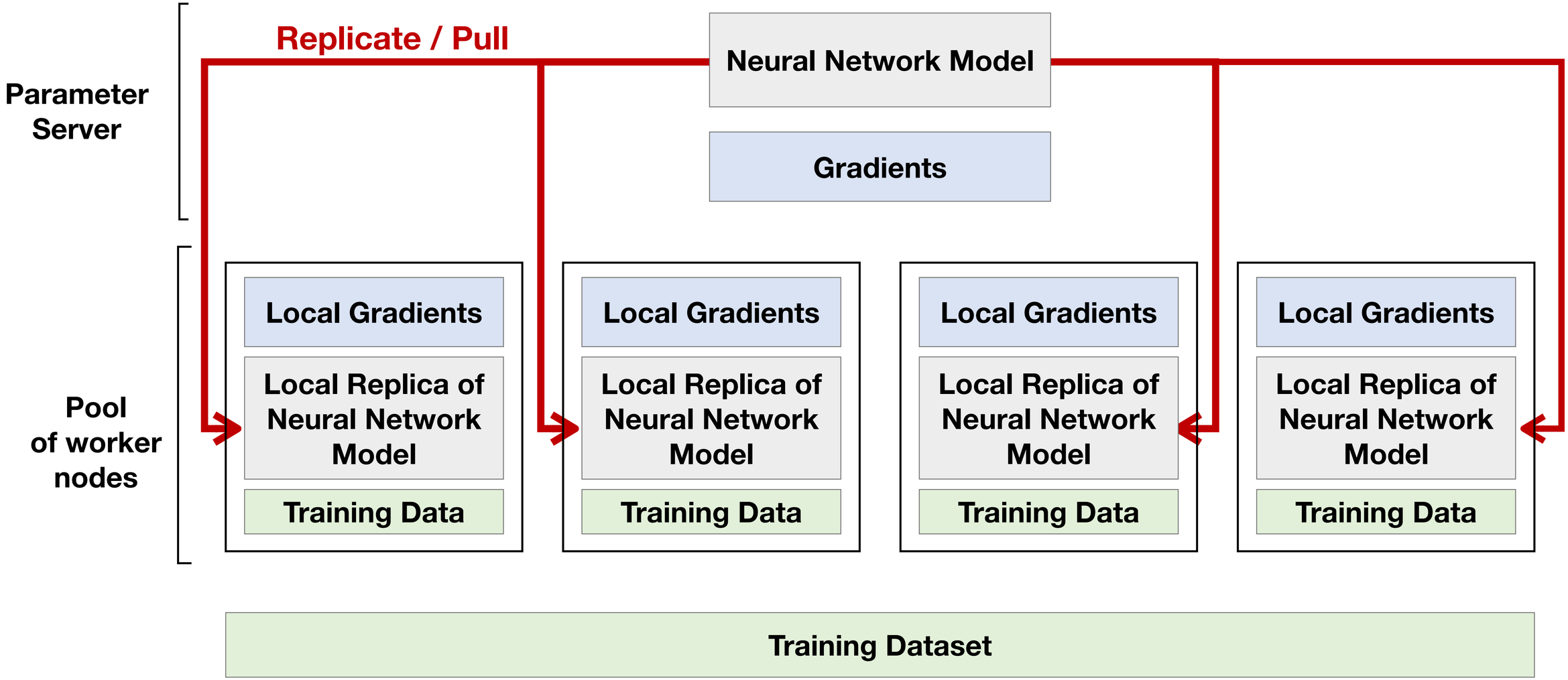
Dive into Data Parallelism

Infrastructure Overview



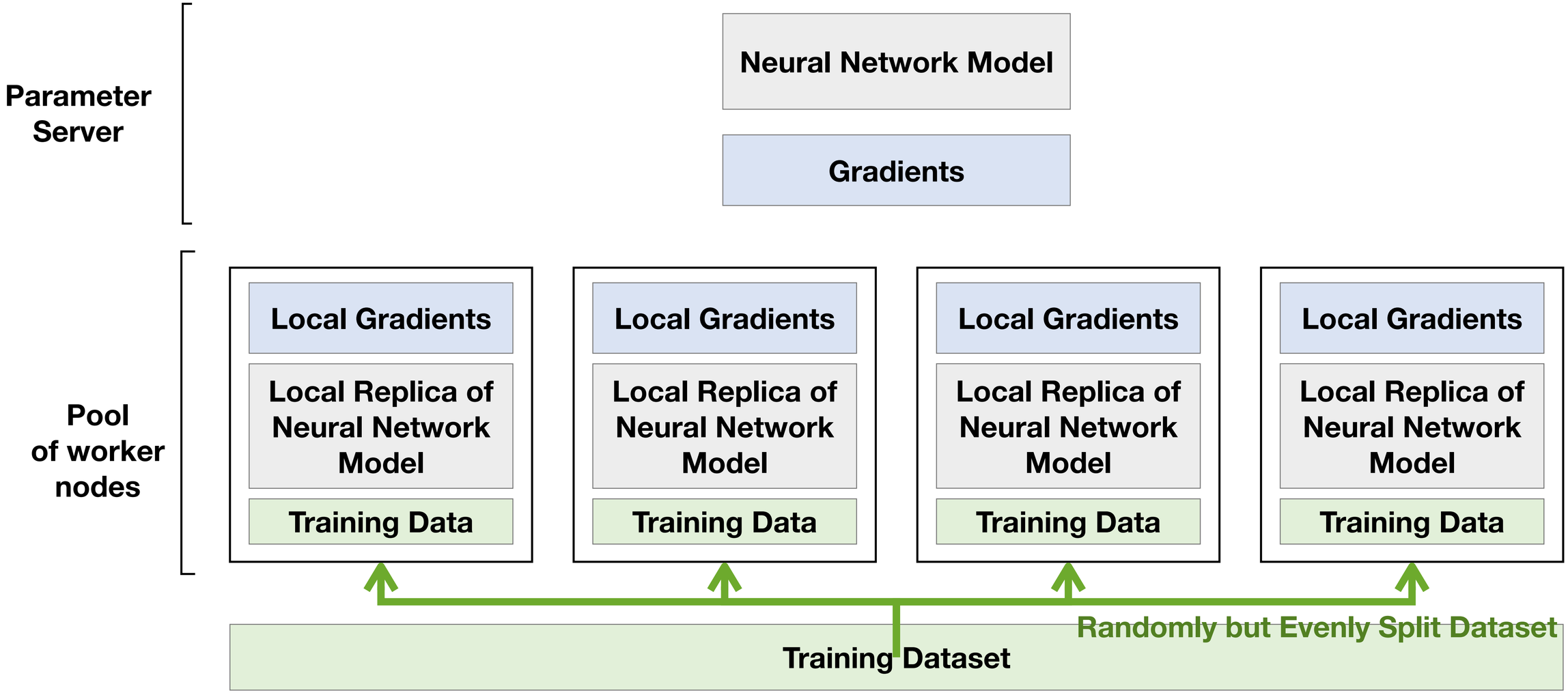
Dive into Data Parallelism

1 - Replicate Models to Each Worker



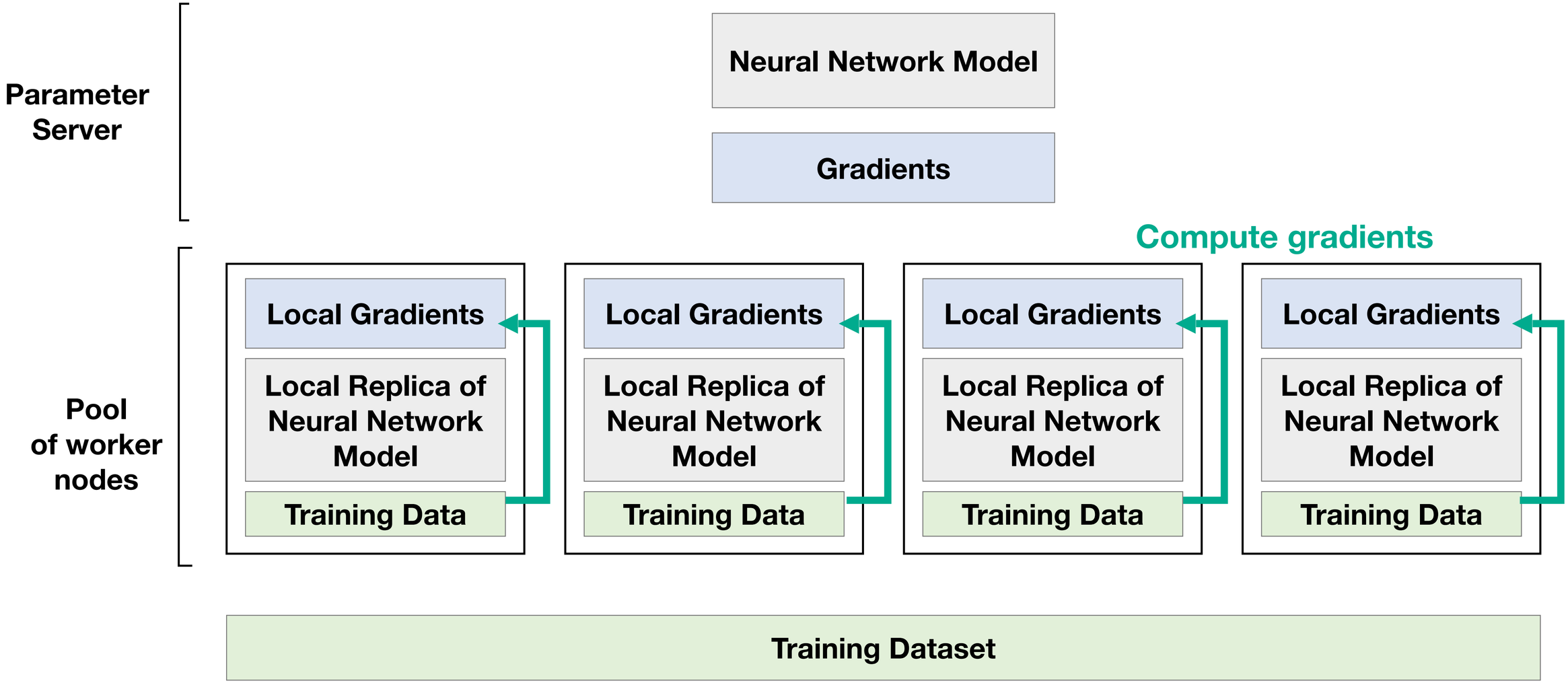
Dive into Data Parallelism

2 - Split Training Data to Workers



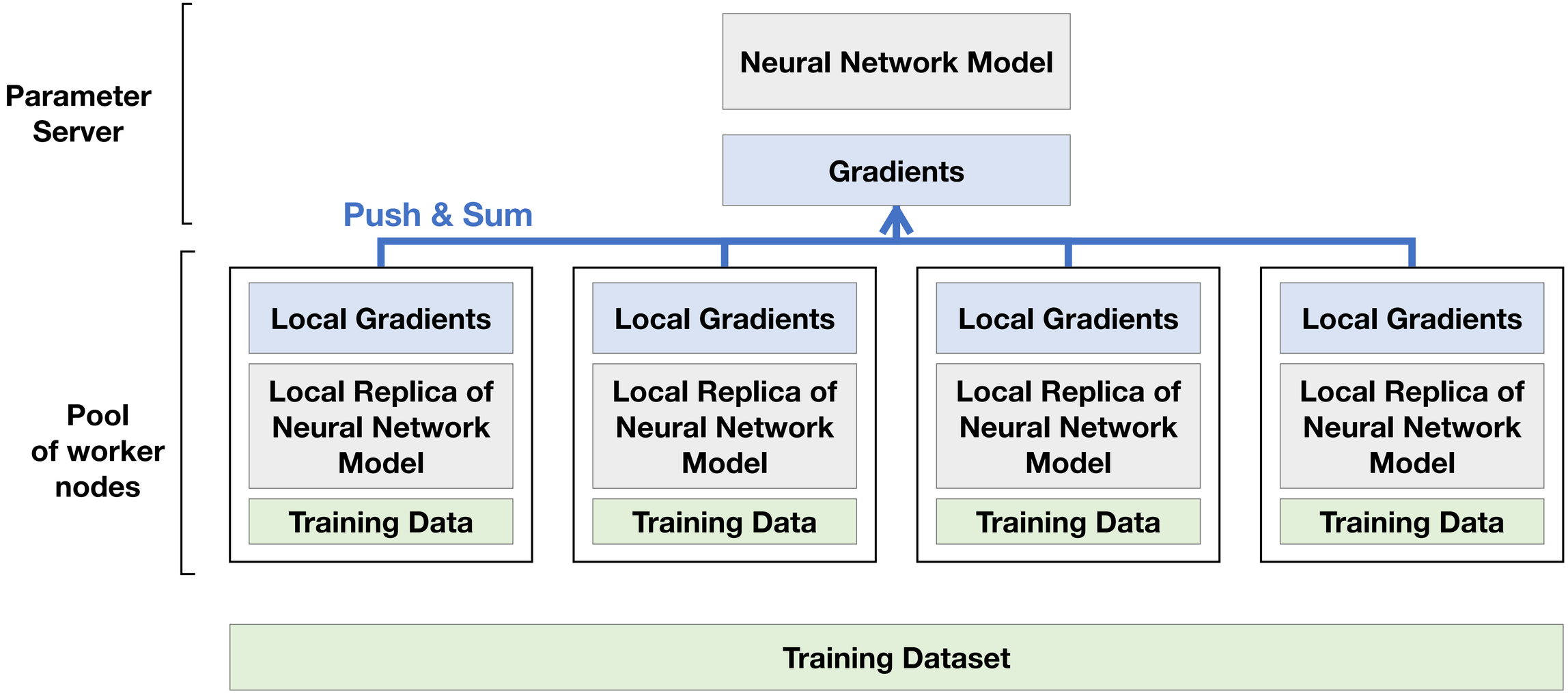
Dive into Data Parallelism

3 - Compute gradients



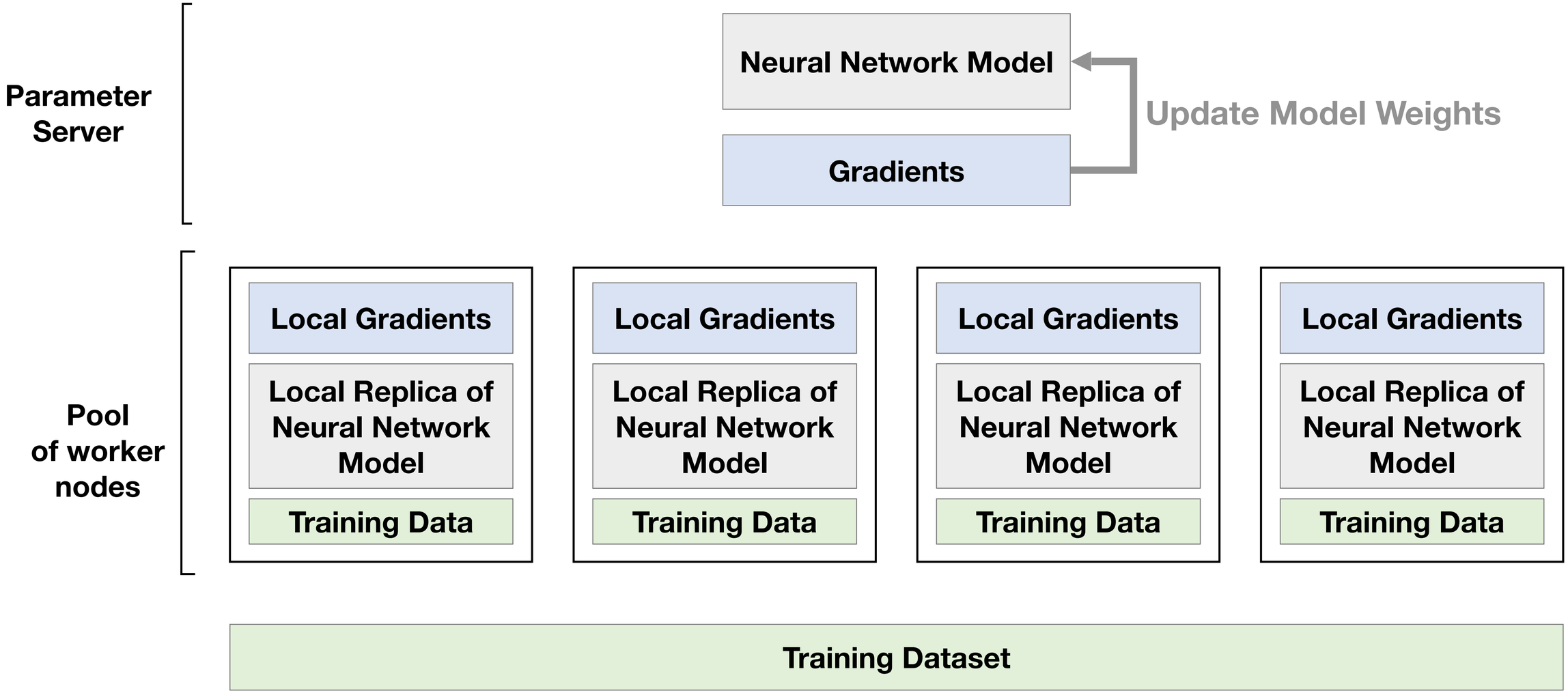
Dive into Data Parallelism

4 - Aggregate and Synchronize Gradients



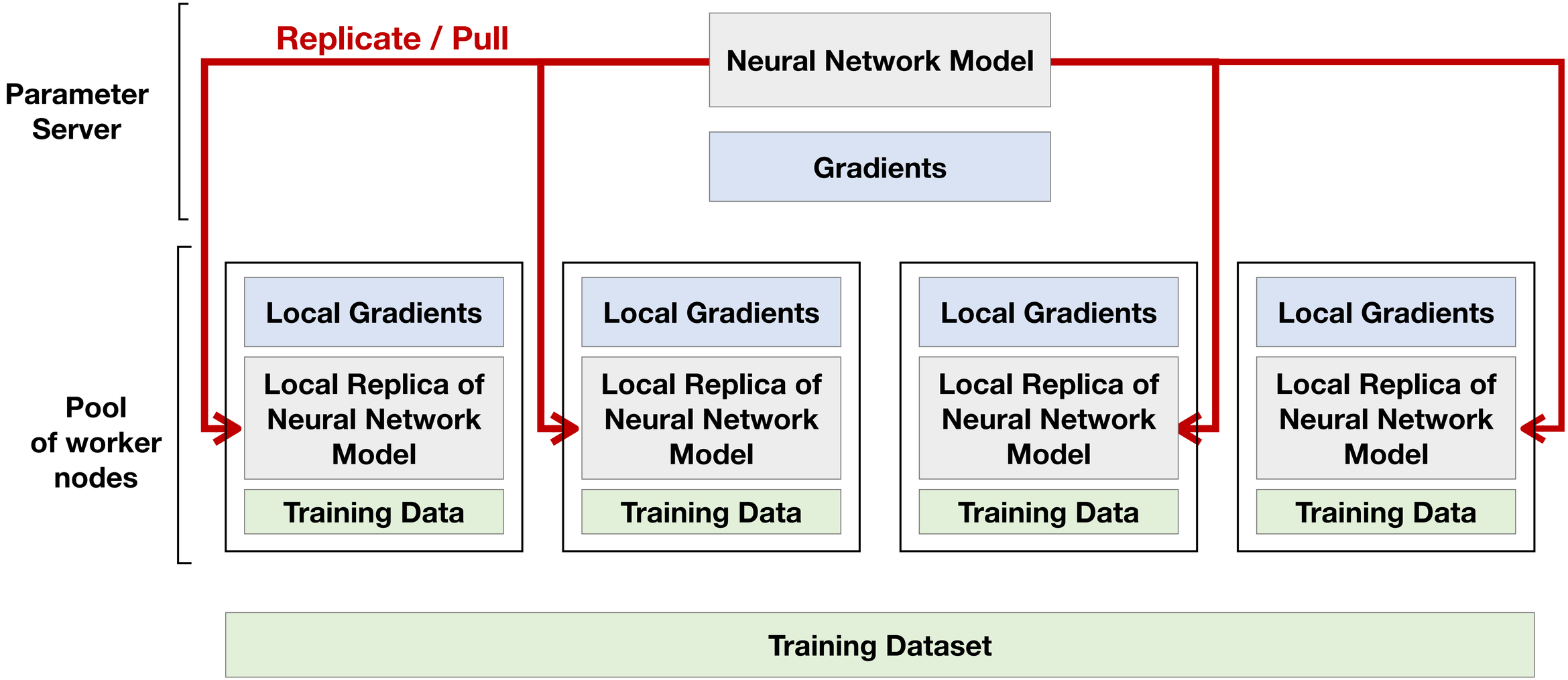
Dive into Data Parallelism

5 - Perform Gradient Step and Update Model Parameters



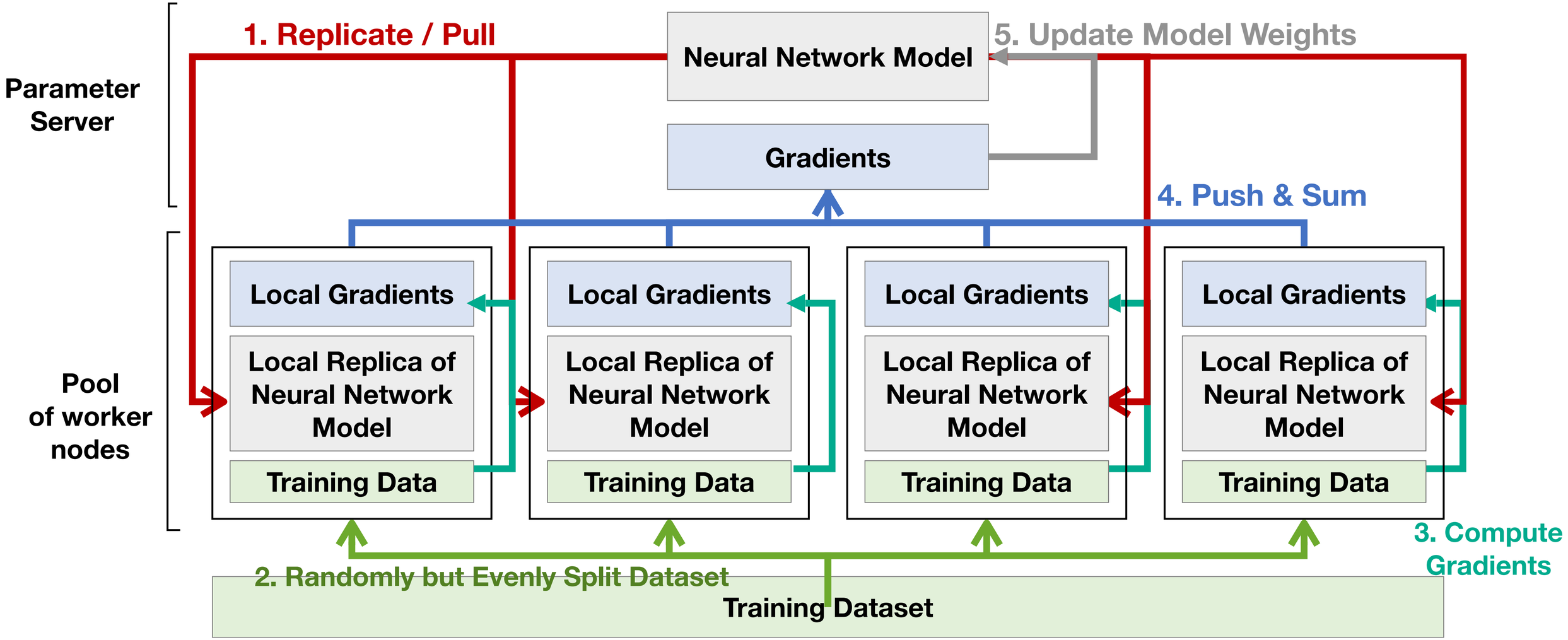
Dive into Data Parallelism

Repeats Previous Steps



Dive into Data Parallelism

Replicate -> Split Data -> Compute Gradients -> Push & Sum -> Update



Compare Single Node and Distributed Training

Single Node Training:

For iteration i in 0 .. T,

1. Sample data from datasets $x_{(i)}, y_{(i)}$
2. Compute gradients $\nabla w_{(i)} = f(x_{(i)}, y_{(i)}; w_{(i)})$
3. Perform gradient step $w_{(i)} = w_{(i)} - \eta \nabla w_{(i)}$

Distributed Training:

For iteration i in 0 .. T,

1. Replicate / pull gradients from parameter server
2. Sample data from datasets $x_{(i,j)}, y_{(i,j)}$
3. Compute gradients $\nabla w_{(i,j)} = f(x_{(i,j)}, y_{(i,j)}; w_{(i,j)})$
4. Push $\nabla w_{(i,j)}$ to parameter server and sum
5. Perform gradient step $w_{(i,j)} = w_{(i,j)} - \eta \overline{\nabla w_{(i)}}$ at parameter server.

Compare Single Node and Distributed Training

Single Node Training:

For iteration i in 0 .. T,

1. Sample data from datasets $x_{(i)}, y_{(i)}$
2. Compute gradients $\nabla w_{(i)} = f(x_{(i)}, y_{(i)}; w_{(i)})$
3. Perform gradient step $w_{(i)} = w_{(i)} - \eta \nabla w_{(i)}$

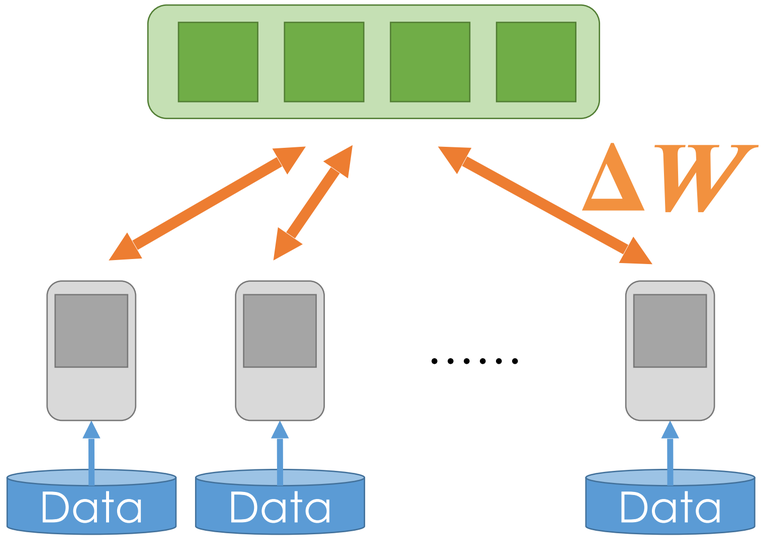
Distributed Training:

For iteration i in 0 .. T,

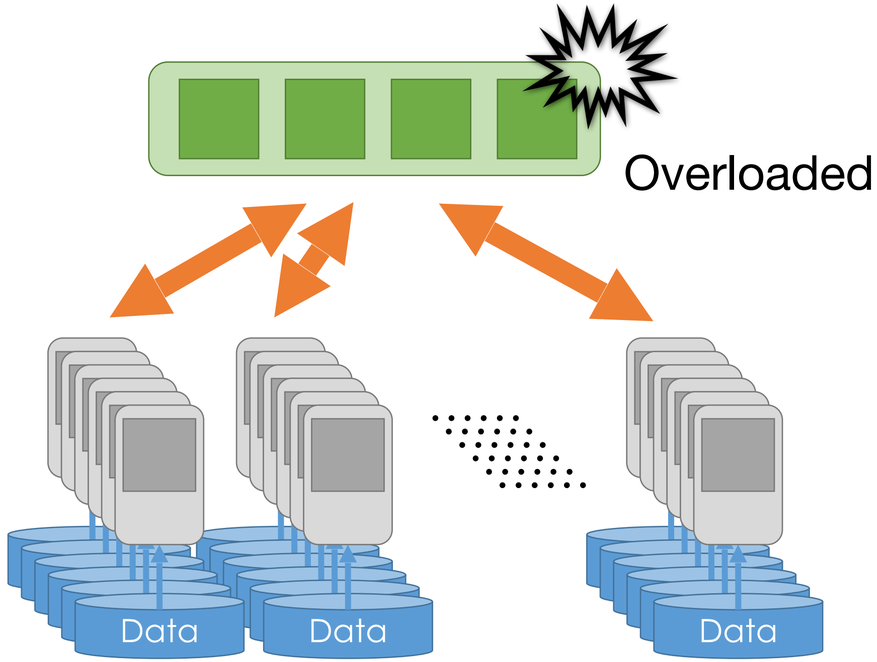
1. Replicate / pull gradients from parameter server
2. Sample data from datasets $x_{(i,j)}, y_{(i,j)}$
3. Compute gradients $\nabla w_{(i,j)} = f(x_{(i,j)}, y_{(i,j)}; w_{(i,j)})$
4. Push $\nabla w_{(i,j)}$ to parameter server and sum
5. Perform gradient step $w_{(i,j)} = w_{(i,j)} - \eta \overline{\nabla w_{(i)}}$ at parameter server.

With **two extra synchronization steps**, we scale training from single node to distributed one.

Problems with Parameter Server



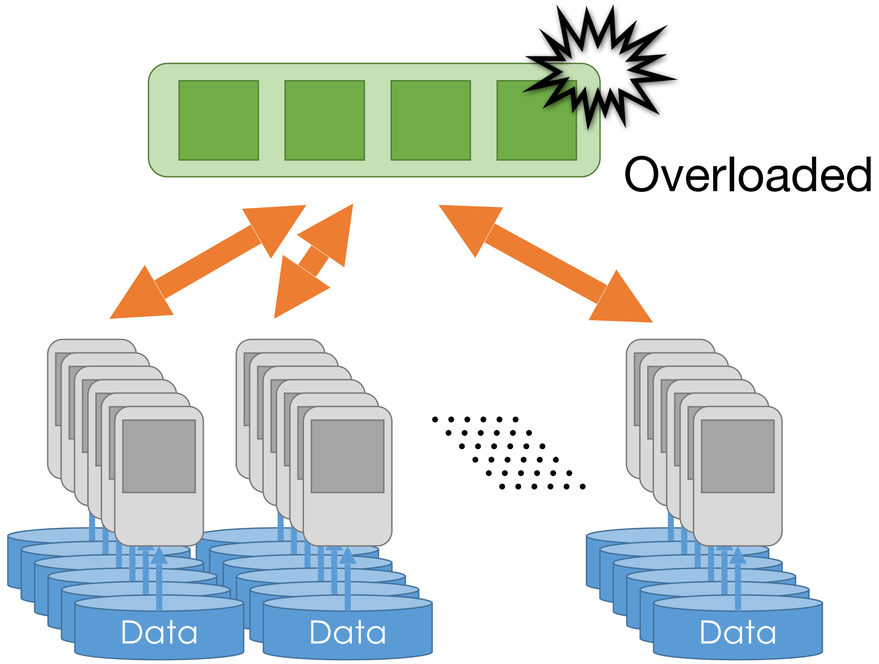
When number of workers is small 😊



When number of workers is large 😭

The bandwidth requirement of parameter server grows linearly w.r.t number of workers.

Problems with Parameter Server



When number of workers is large 😭

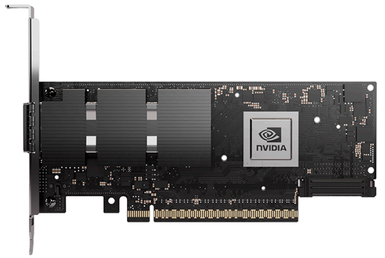
Training ResNet50 on V100

ResNet50: 24.4M Params = 97.5MB Storage

Forward + Backward: 3 iterations per second (bs=32, 2080ti)

If we train with 256 nodes, the bandwidth requirements of parameter server is

256 * 3 * 97.5 = **73.1 GB/s!**



Mellanox infiniband ConnectX-5:

12.5GB/s (not enough)

Can we perform distributed training **without** centralized server?

Section 4: Distributed Communication Primitives

Basic Communication Schemes and Ring-AllReduce

Distributed Communication

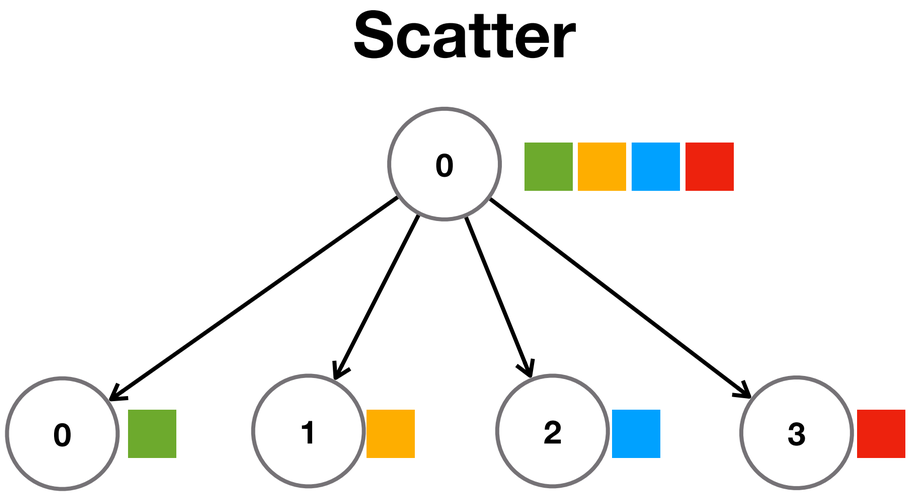
Point-to-Point: Send and Recv

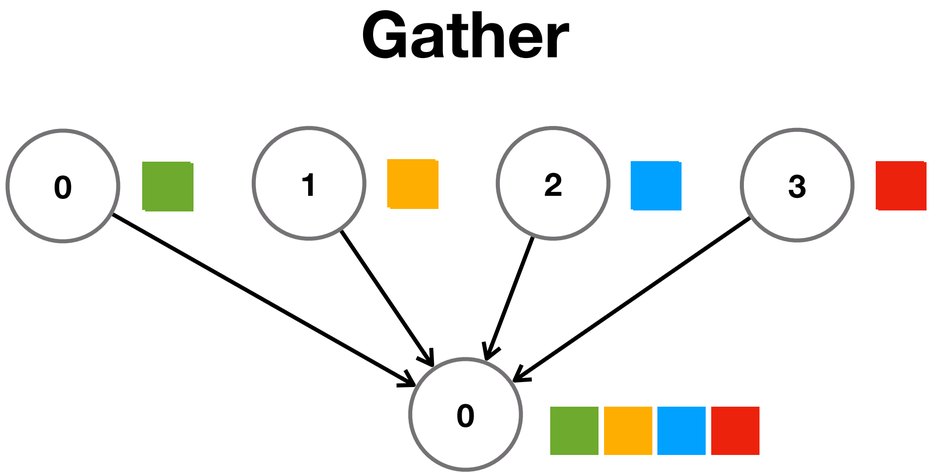
Point-to-point communication: transfer data from one process to another

- Send & Receive are the most common distributed communication schemes.
- Implemented in Socket / MPI / Gloo / NCCL

Distributed Communication

Collective Communication: Scatter and Gather





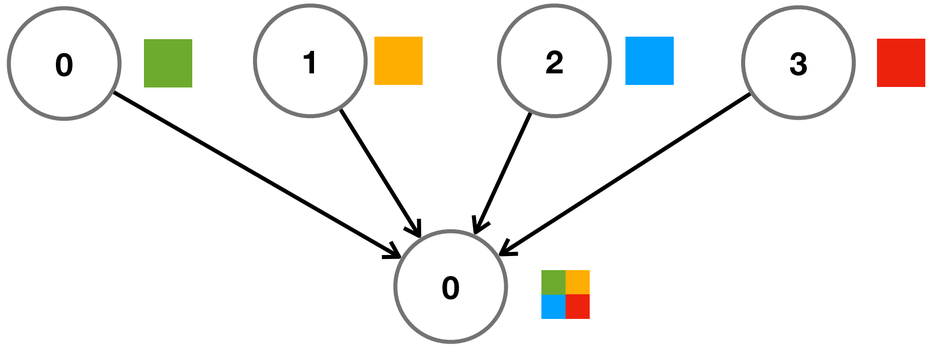
Collectives communication: a type of operations performed across all worker.

- Scatter: send a tensor to all other workers
- Gather: receive a tensor from all other workers

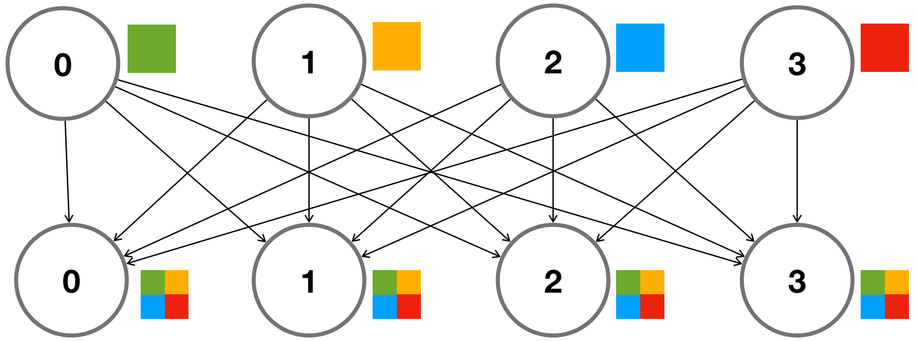
Distributed Communication

Collective Communication: Reduce and All Reduce

Reduce



All-Reduce

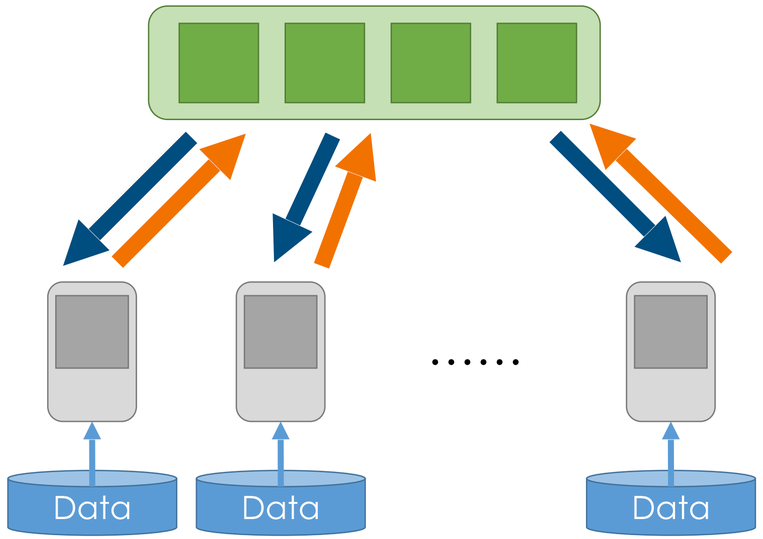


Collectives performs communication patterns across all worker.

- Reduce: similar to gather, but averaging / summing during aggregation.
  - [1] [2] [3] [4] —(gather)—> [1,2,3,4]
  - [1] [2] [3] [4] —(reduce)—> [1 + 2 + 3 + 4] = [10]
- All-Reduce: perform Reduce on all workers

Distributed Communication

The communication schemes used in Parameter Server (PS)

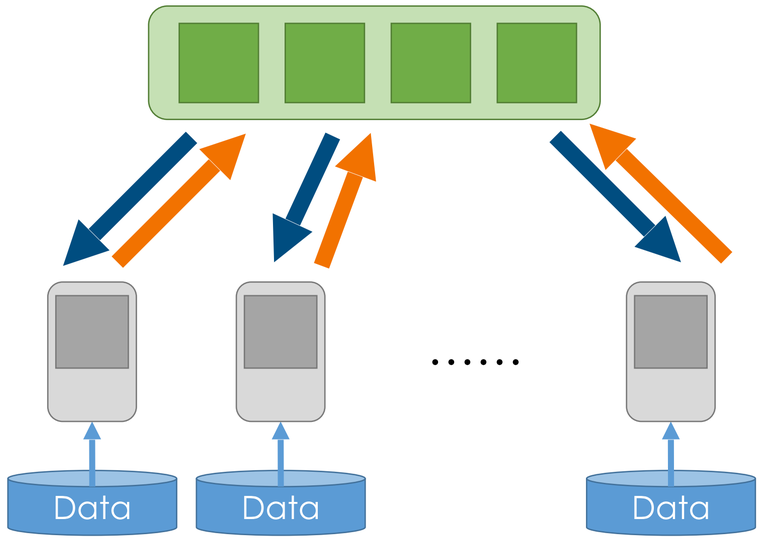


- Which type of communication scheme is used?
  - **Replicate & Pull**: Send
    - Parameter Server to all workers
  - **Push & Sum**: Reduce
    - Average from all workers to Parameter Server
- What is the bandwidth requirements on each node?
  - **Replicate & Pull**:
    - Worker: O(1)
    - Parameter Server: O(N)
  - **Push & Sum**:
    - Worker: O(1)
    - Parameter Server: O(N)

Note: N is the number of training workers.

Distributed Communication

The communication schemes used in Parameter Server (PS)



- **Replicate & Pull**:
  - Worker: O(1)
  - Parameter Server: O(N)

- **Push & Sum:**
  - Worker: O(1)
  - Parameter Server: O(N)

The bandwidth of central parameter server grows linearly w.r.t #num of workers, which can be a **bottleneck** when there are more machines.

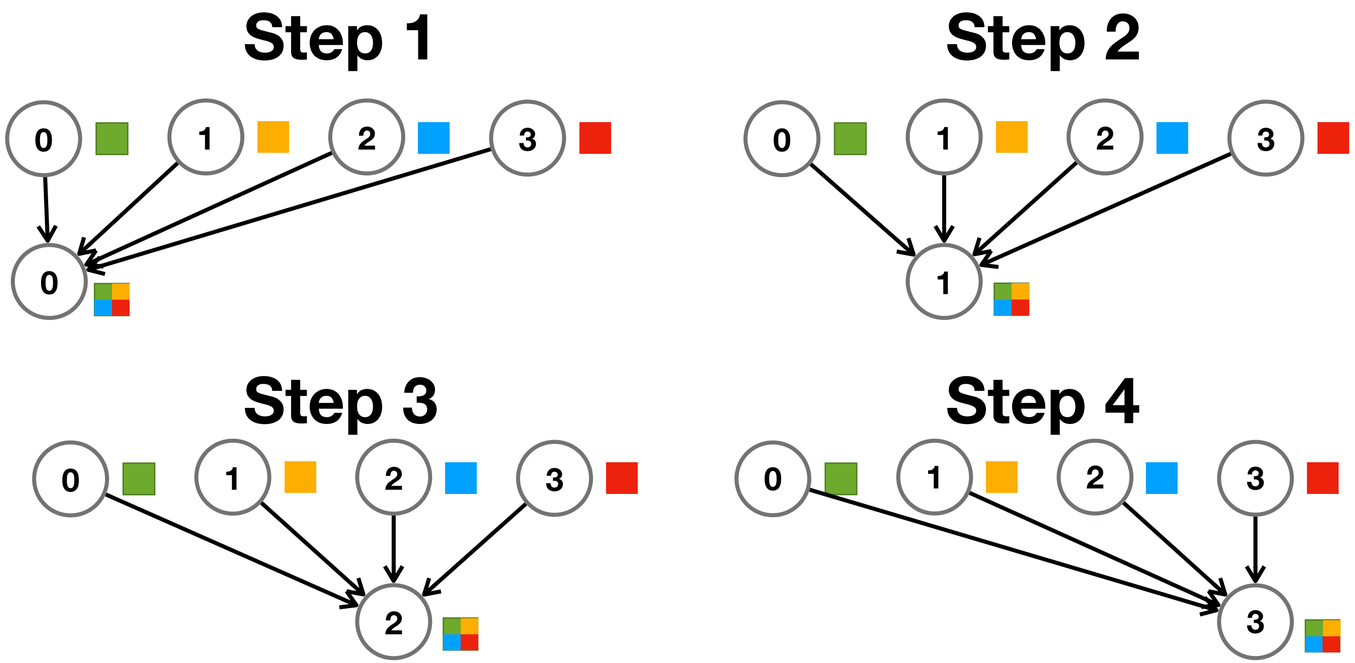
Can we perform the aggregation without a central server?

All-Reduce!

Note: N is the number of training workers.

Distributed Communication

Naive All-Reduce Implementation - Sequential




```
Pseudocode:
_________________
For i:=0 to N:
  Allreduce(work[i])
```

Time: O(N)

Bandwidth: O(N)

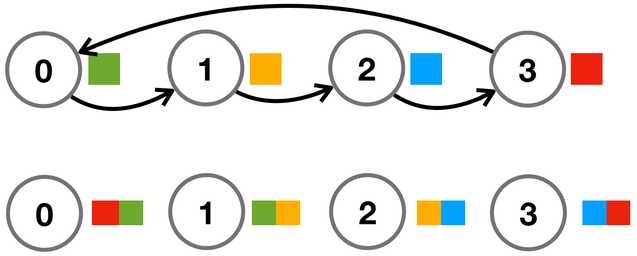
Each step performs a **SINGLE** Reduce operation

Note: N is the number of training workers.

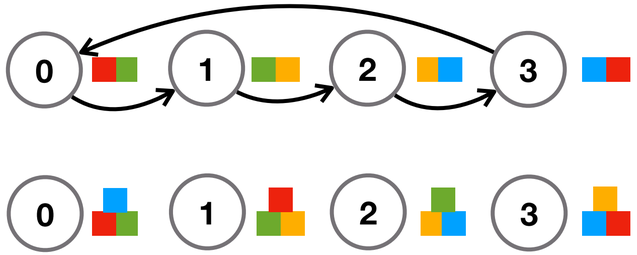
Distributed Communication

Better All-Reduce Implementation - Ring

Step 1



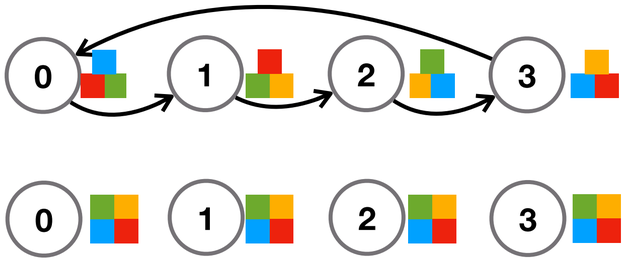
Step 2



Pseudocode:

```
For i:=0 to N:
   send(src=i, dst=(i+1)%N, data=worker[i])
   obj = recv(src=(i+1)%N, dst=i)
   worker[i] = merge(obj, worker[i])
```

Step 3



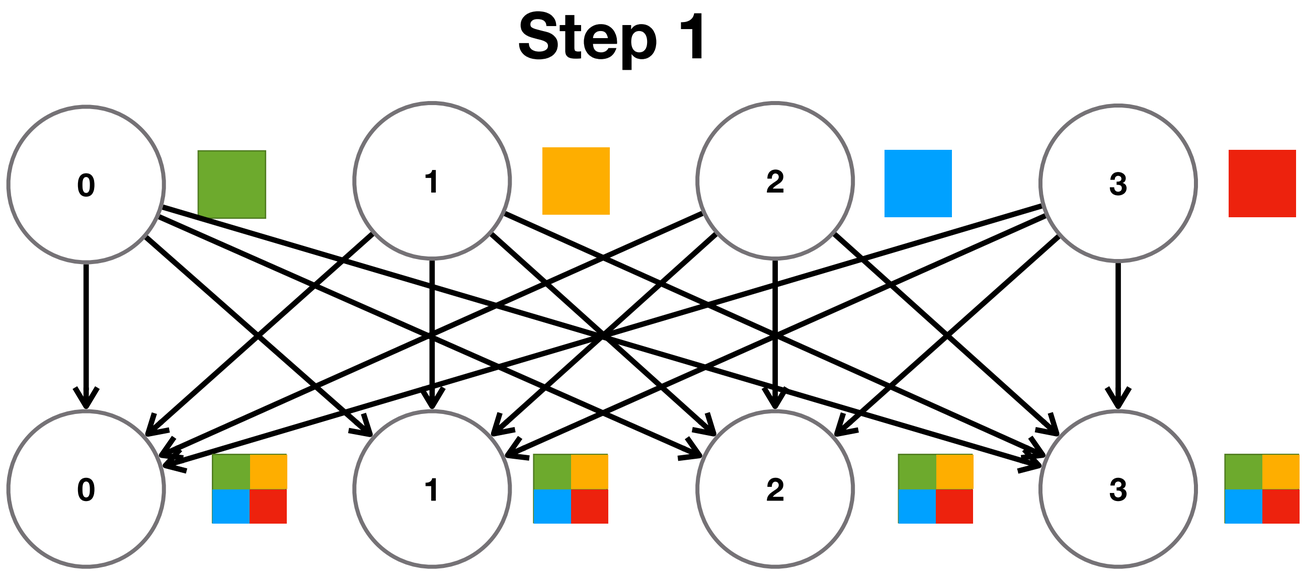
Time: O(N)

Bandwidth: O(1) **<- reduced**

Each step performs a **SINGLE** send and merge.

Distributed Communication

Naive All-Reduce Implementation - Parallel Reduce




```
Pseudocode:
_________________
Parallel for i:=0 to N:
  Allreduce(work[i])
```

Time: O(1) **<- improved**

Bandwidth: O(N^2) **<- worse**

Perform **ALL** reduce operations simultaneously.

Distributed Communication

Compare Different Distributed Aggregation

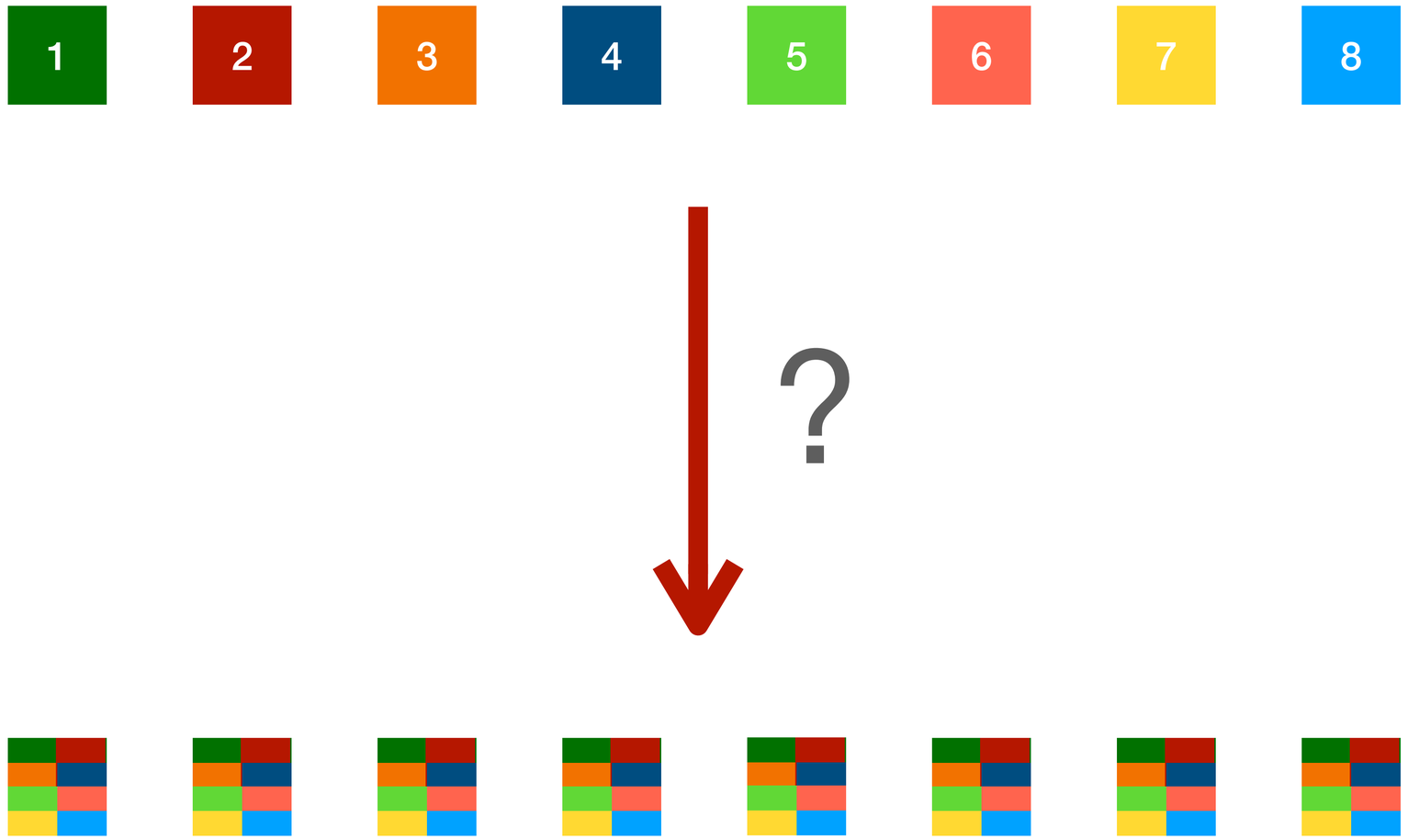
	Time	Peak Node Bandwidth	Total Bandwidth
Parameter Server	O(1)	O(N)	O(N)
All-Reduce - Sequential	O(N)	O(N)	O(N)
All-Reduce - Ring	O(N)	O(1)	O(N)
All-Reduce - Parallel	O(1)	O(N)	O(N^2)

Can we combine the advantages of Ring and Parallel?

Sure

Distributed Communication

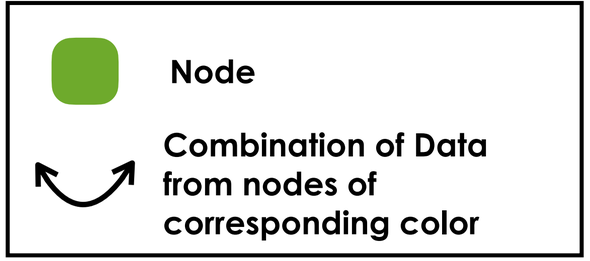
Recursive Halving All Reduce



[1] Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. "Optimization of collective communication operations in MPICH."

Distributed Communication

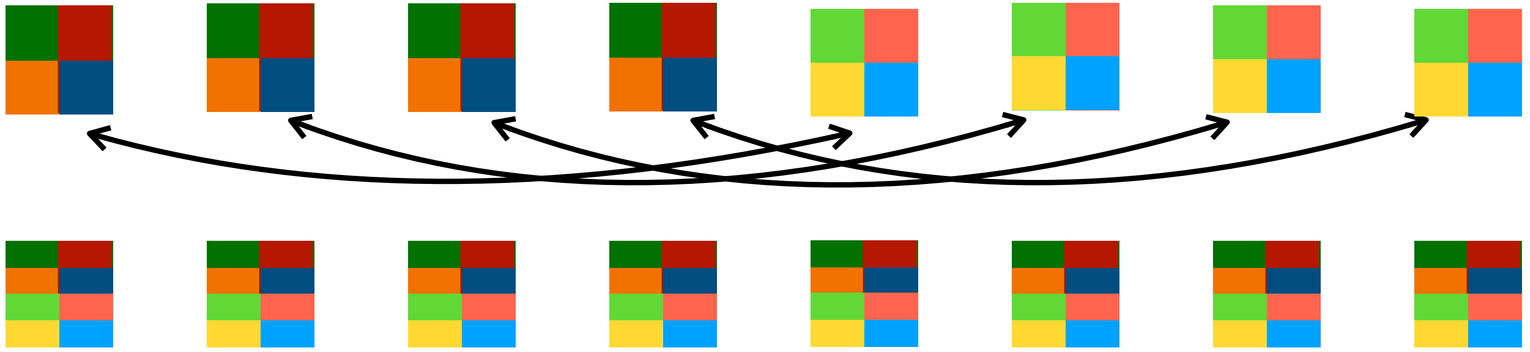
Recursive Halving All Reduce



Step 1 - Each node exchanges with neighbors with offset 1

Step 2 - Each node exchanges with neighbors with offset 2

Step 3 - Each node exchanges with neighbors with offset 4



For N workers, AllReduce finish in **log(N)** steps.

[1] Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. "Optimization of collective communication operations in MPICH."

Distributed Communication

All Reduce Implementations Comparison

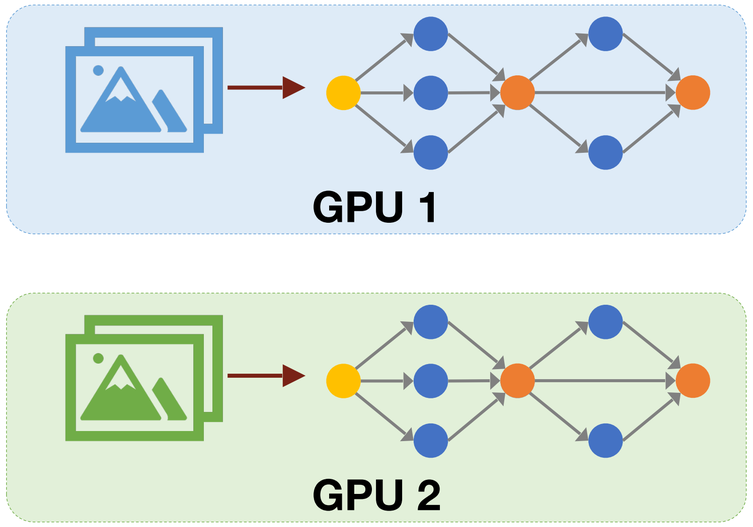
	Time	Peak Node Bandwidth	Total Bandwidth
Parameter Server	O(1)	O(N)	O(N)
All-Reduce - Sequential	O(N)	O(N)	O(N)
All-Reduce - Ring	O(N)	O(1)	O(N)
All-Reduce - Parallel	O(1)	O(N)	O(N^2)
All-Reduce - Recursive Halving	O(lgN)	O(1)	O(N)

AllReduce with proper implementations reduce the peak bandwidth from O(N) to O(1) with little time overhead.

Section 5: Distributed Training with Model Parallelism

Why model parallelism and when to use it?

Data Parallelism Cannot Train Large Models



Though model parallelism has better device utilization, if train a super-large model (e.g., GPT-3)

175B * 16 Bits
= 350GB Memory

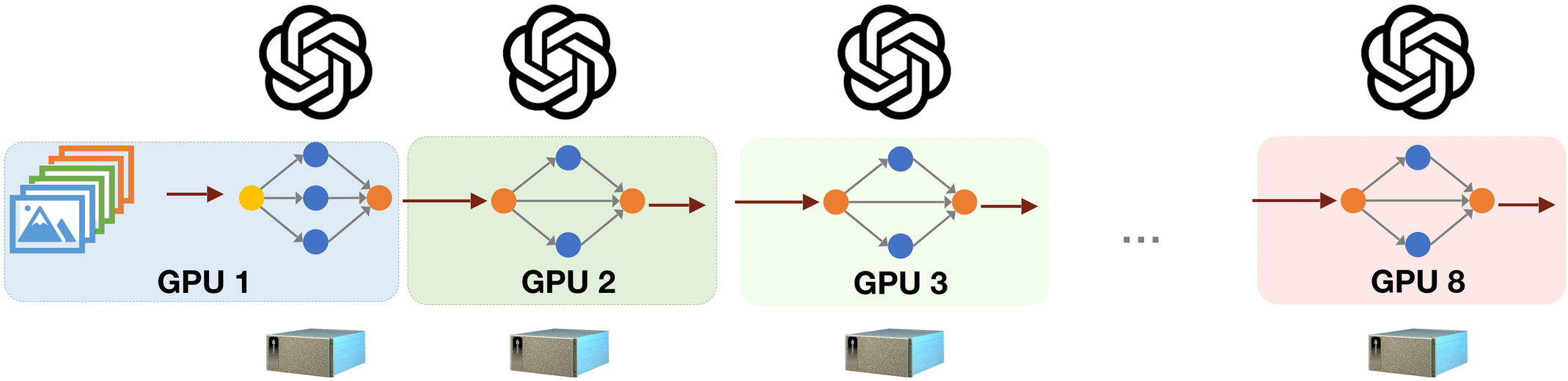
>

Nvidia A100 80GB

Even the best GPU **CANNOT** fit the model into memory!

Model Parallelism Designed for Large Model Training

In order to fit training into hardware, instead of splitting the data, model parallelism split the model

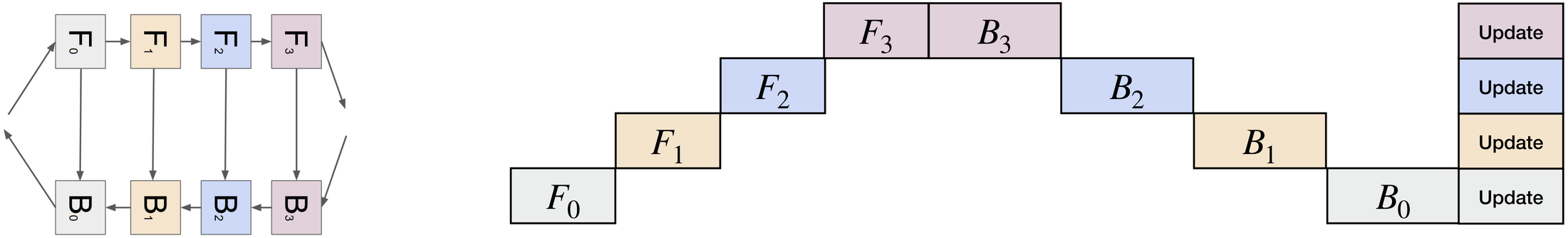


350GB / 8 cards = 43.75G < 80G

With model parallelism, large ML models can be placed and trained on GPUs.

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

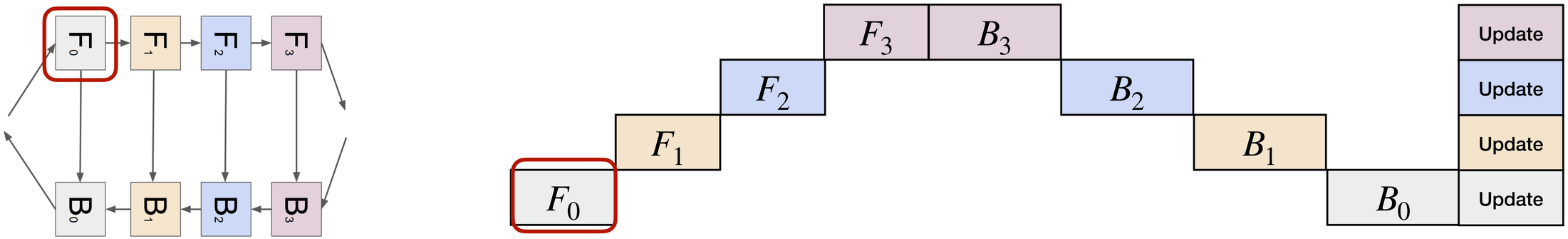
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

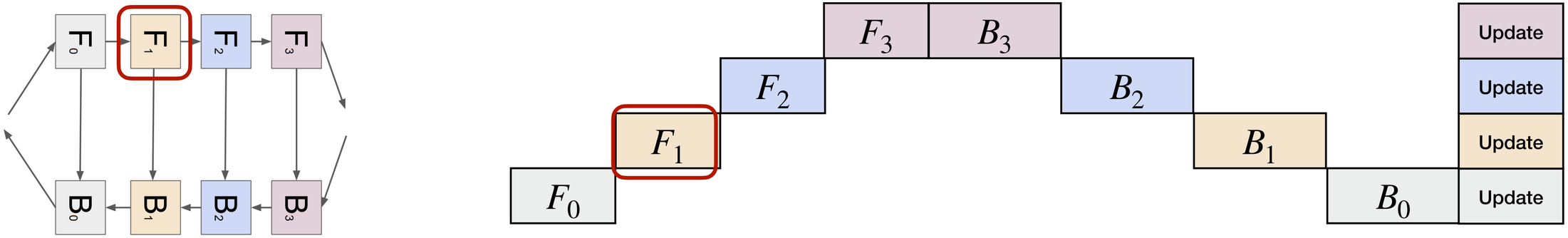
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

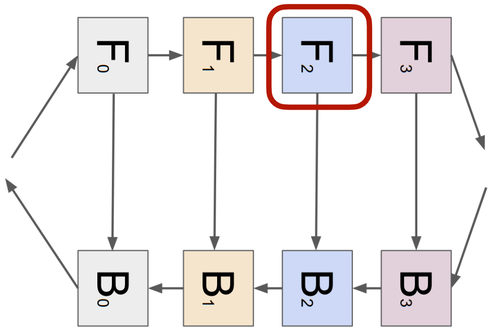
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

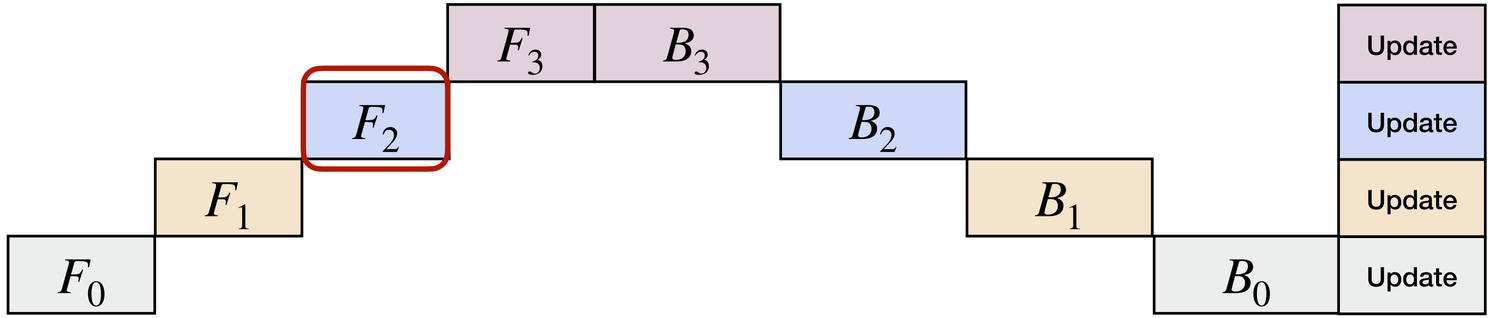
Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow



(b). Training timeline

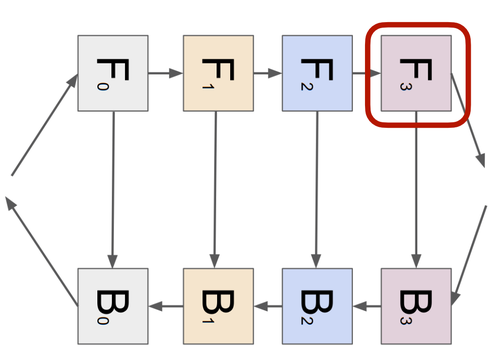
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

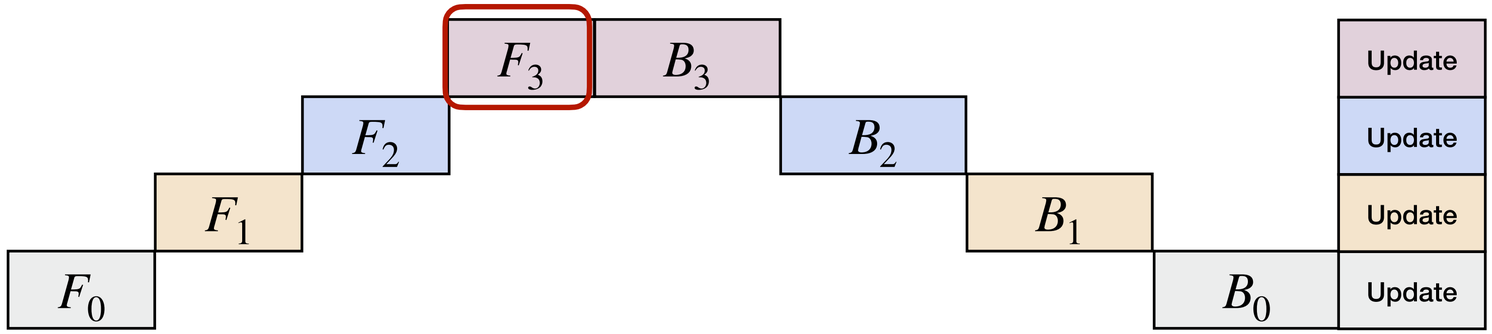
Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow



(b). Training timeline

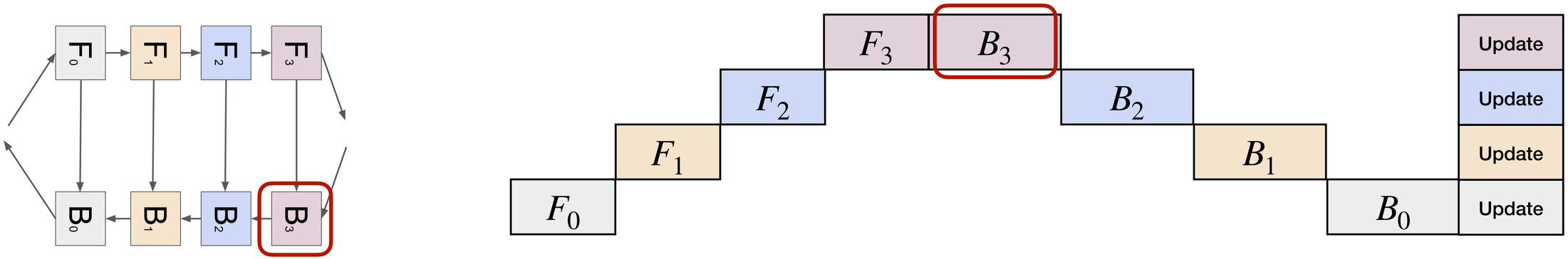
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

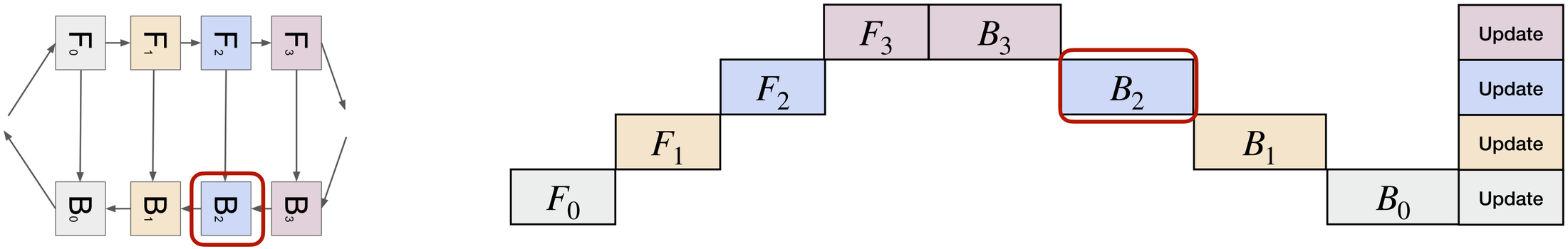
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

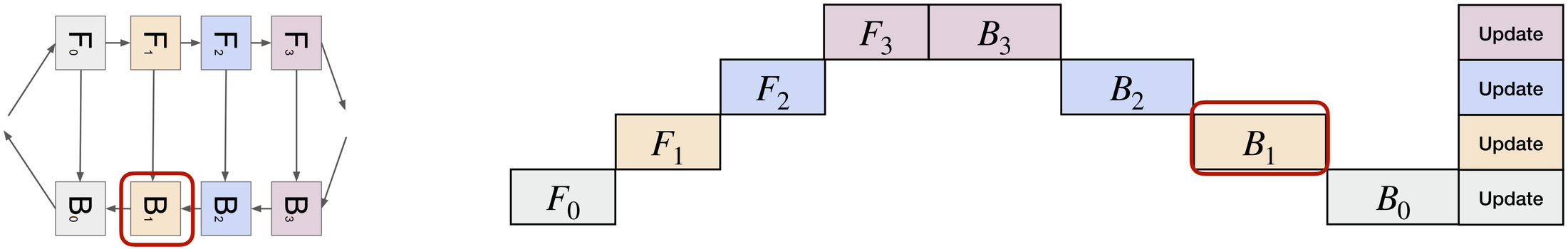
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

(b). Training timeline

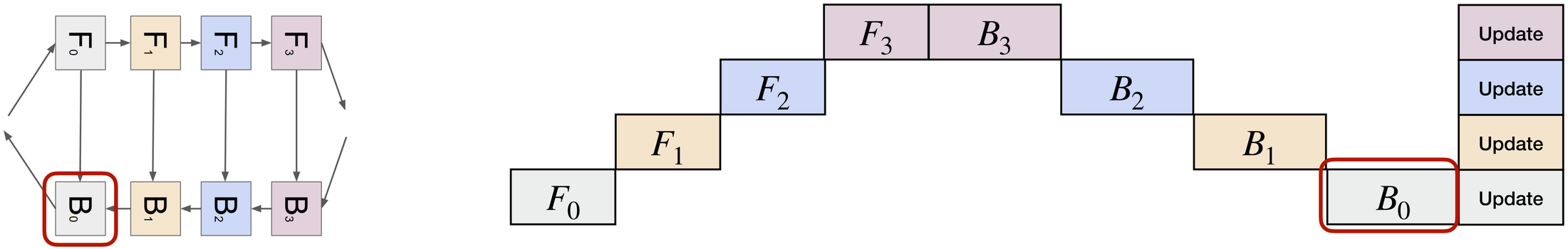
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow

Naive Implementation



(a). Training data flow

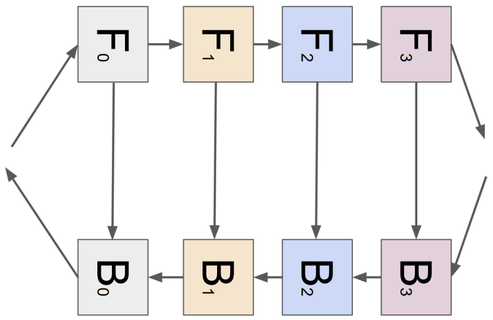
(b). Training timeline

F: Forward B: Backward. Train a 4 layer network with model parallelism.

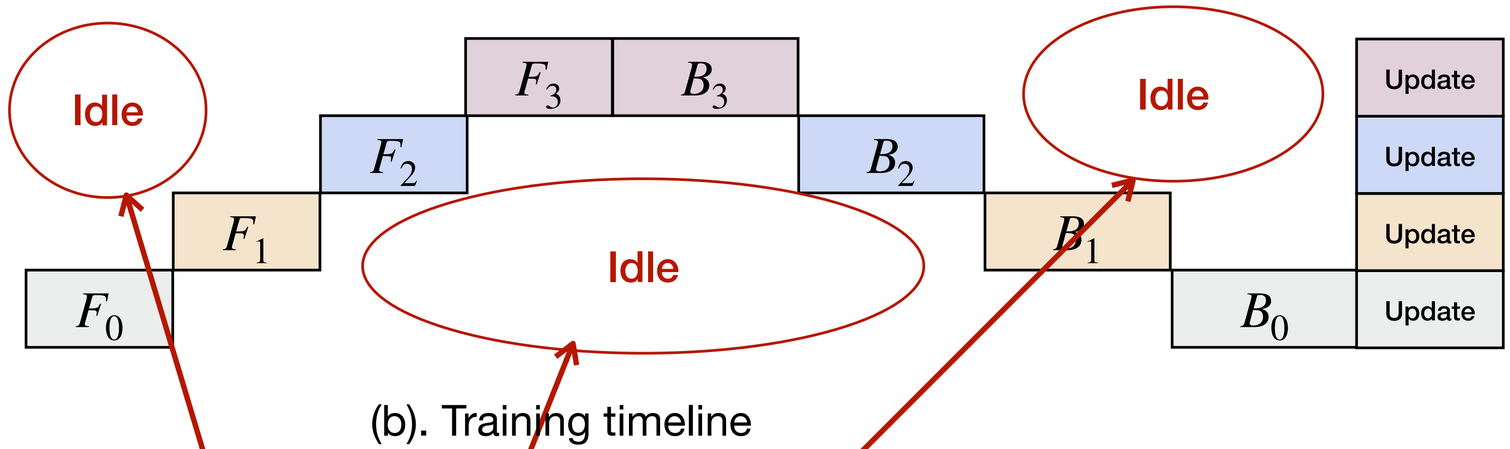
Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from *GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism*

Model Parallelism Workflow



(a). Training data flow



(b). Training timeline

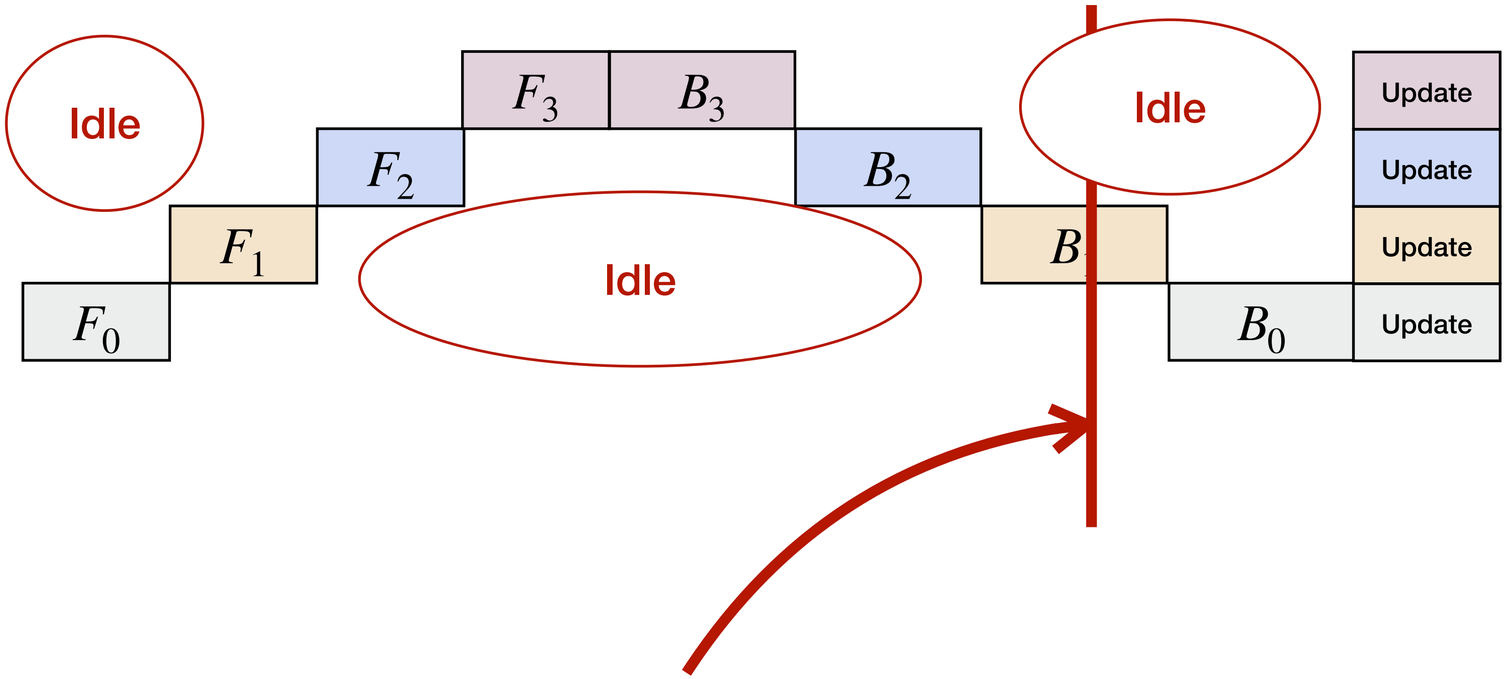
F: Forward B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

But accelerators are significantly **under-utilized** during model parallelism!

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Naive Model Parallelism Suffers from Utilization



Only one device is computing at a time and others are waiting for it.

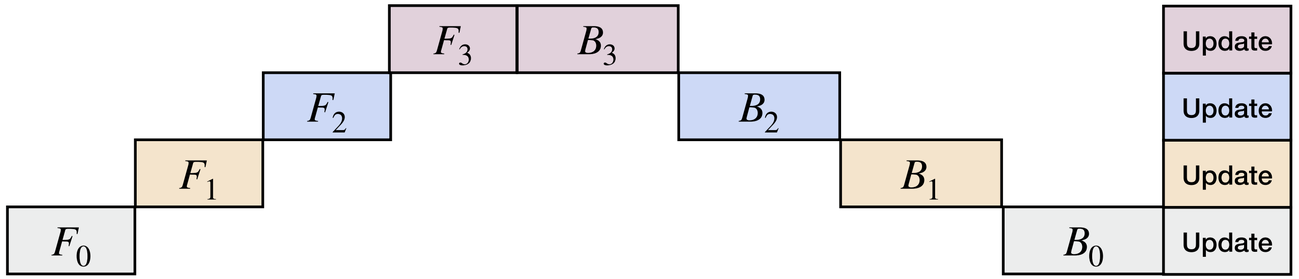
Theoretical utilization: 25% (**low!**)

Usual data parallelism utilization: ~75%

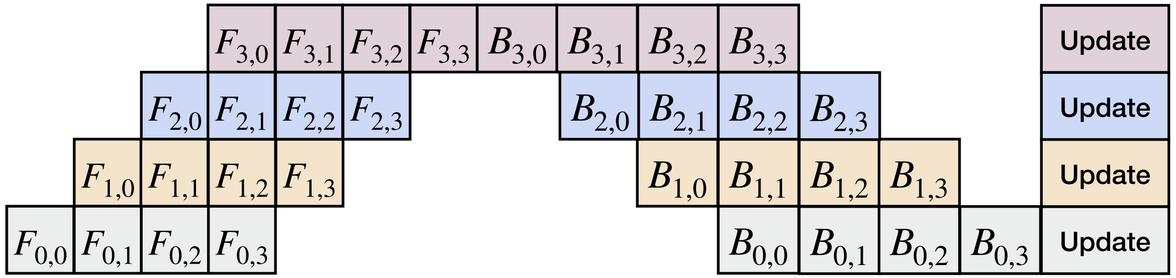
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Pipeline Parallelism

Gpipe: *Easy Scaling with Micro-Batch Pipeline Parallelism*



(a). Naive model parallelism



(b). Pipeline parallelism

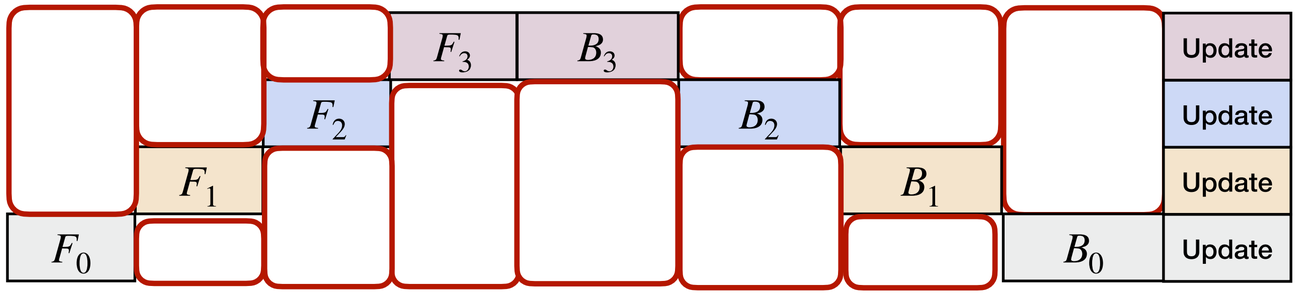
- Split a single batch to micro batches
  - [16, 10, 512] ->
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]

- Motivation: model parameters are not changed during computation within a batch, thus we can pipeline computation and communication

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Pipeline Parallelism

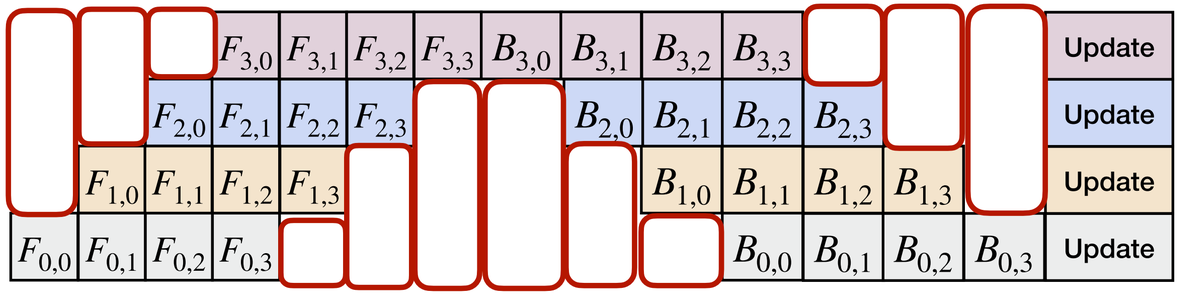
Micro-batch improvs the device utilization



(a). Naive model parallelism

Utilization: (25%)

The more chunks (#num of micro batches), the higher of device utilization.



(b). Pipeline parallelism

Utilization: **57%** (2.5x improvement)

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

GPipe: Performance analysis

Efficient study on TPU

TPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 1$	1	1.13	1.38	1	1.07	1.3
$M = 4$	1.07	1.26	1.72	1.7	3.2	4.8
$M = 32$	1.21	1.84	3.48	1.8	3.4	6.3

- Normalized training throughput using GPipe with different # of partitions ***K*** and different # of micro-batches ***M*** on TPUs.
- The more #num of partitions, the more #num of micro-batches, the higher the training throughput.
- With reasonable ***M*** and ***K***, there is an almost **linear speedup** when training transformer.

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Beyond Model Parallelism

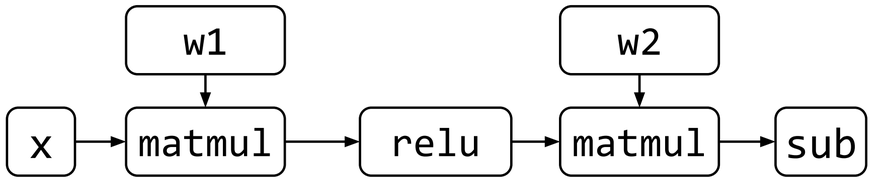
Motivation

- When a model is too large to be put on GPU
  - Split the model and use model parallelism!
- But what if a layer is too large for the memory?
  - Split the layer and we get tensor parallelism!

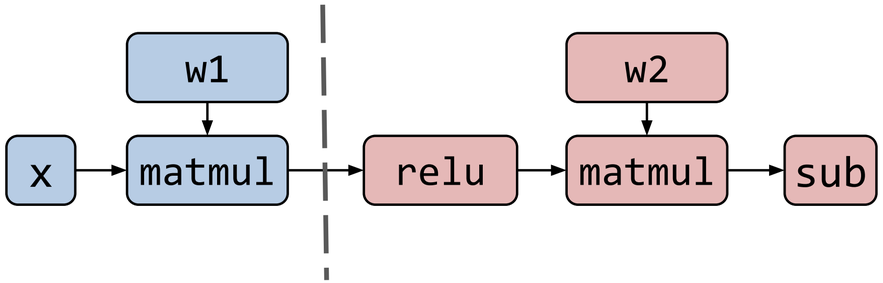
Beyond Model Parallelism

Motivation

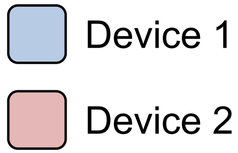
- When a model is too large to be put on GPU
  - Split the model and use model parallelism!
- But what if a layer is too large for the memory?
  - Split the layer and we get tensor parallelism!

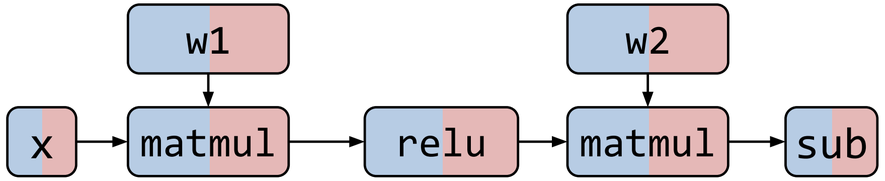


(a) Computation Graph



(b) Model Parallelism

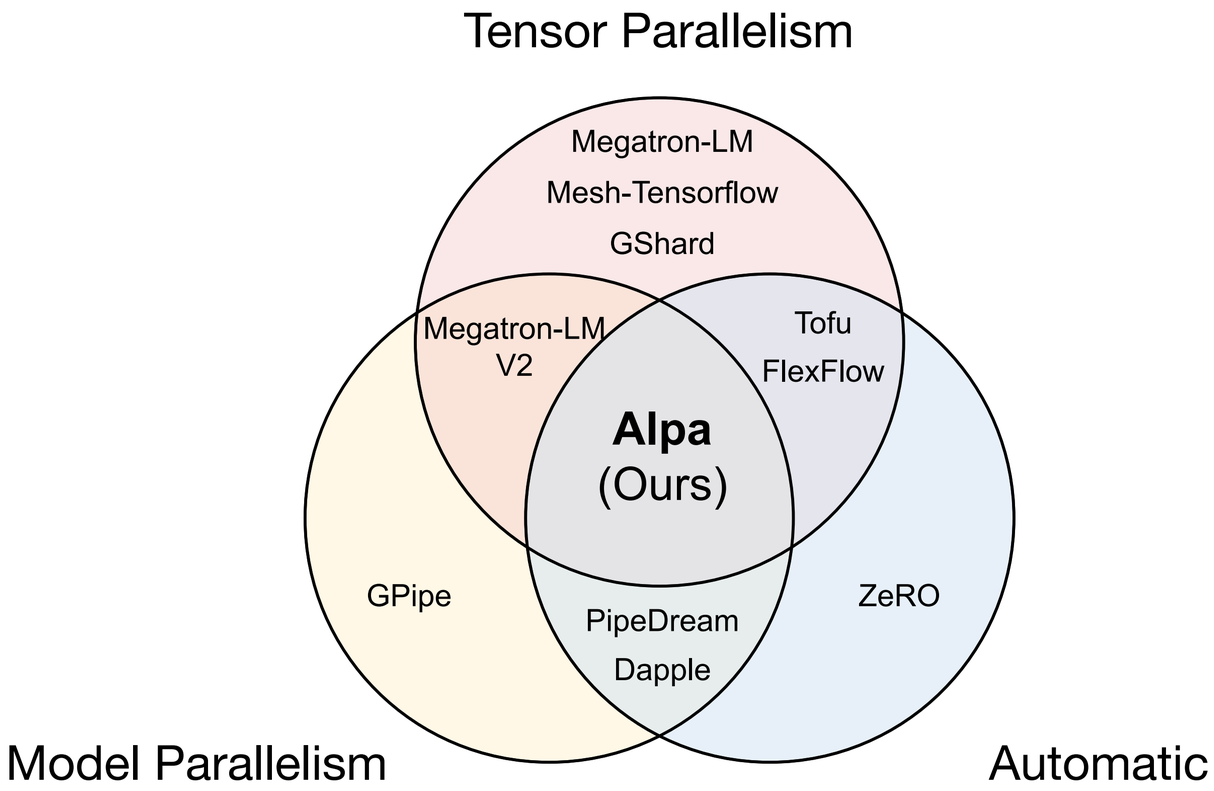


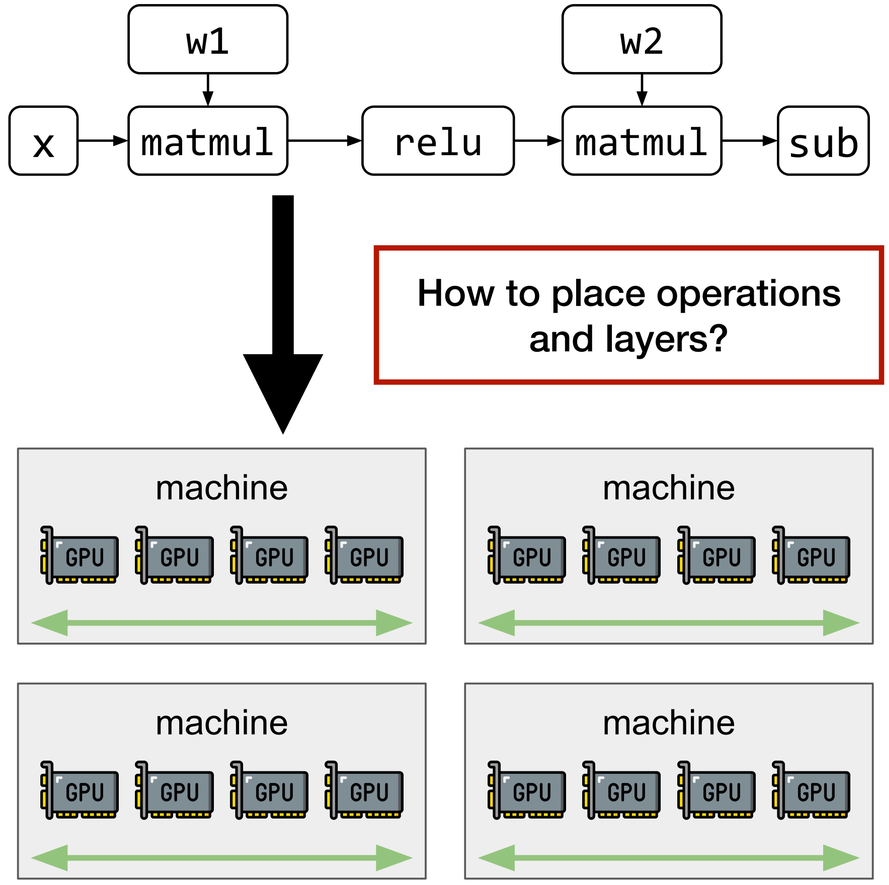


(c) Tensor Parallelism

How to find the best parallel strategies?

Alpa: Automating Inter- and Intra-Operator Parallelism



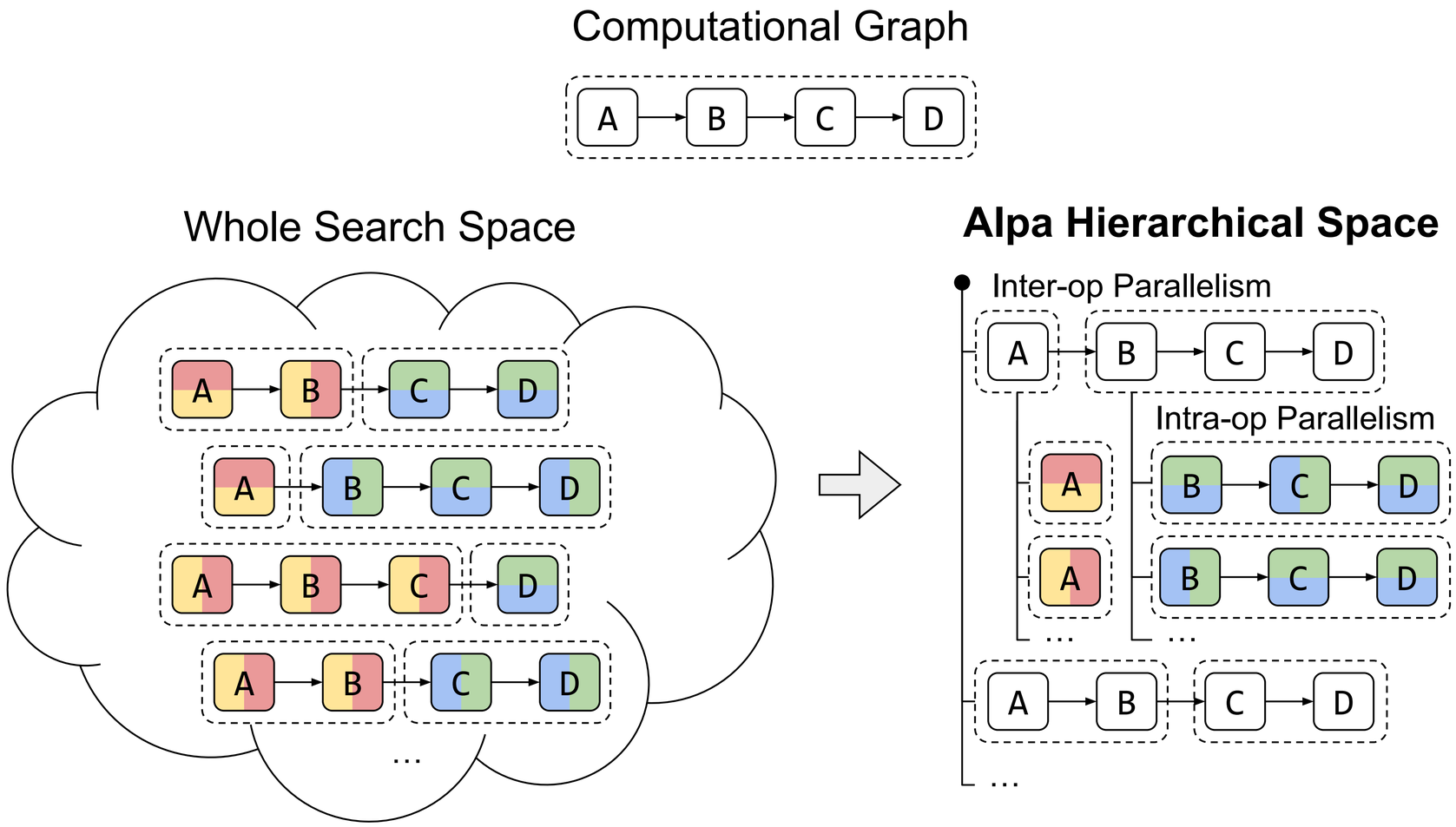


Apla utilizes different degrees of parallelism.

Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning [Zheng et al. 2022]

Alpa: A Unified Compiler for Distributed Training

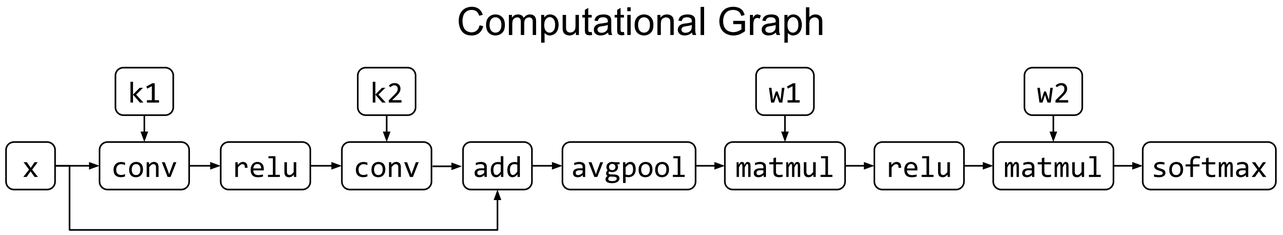
Define the Search Space for Parallel Strategies



Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning [Zheng et al. 2022]

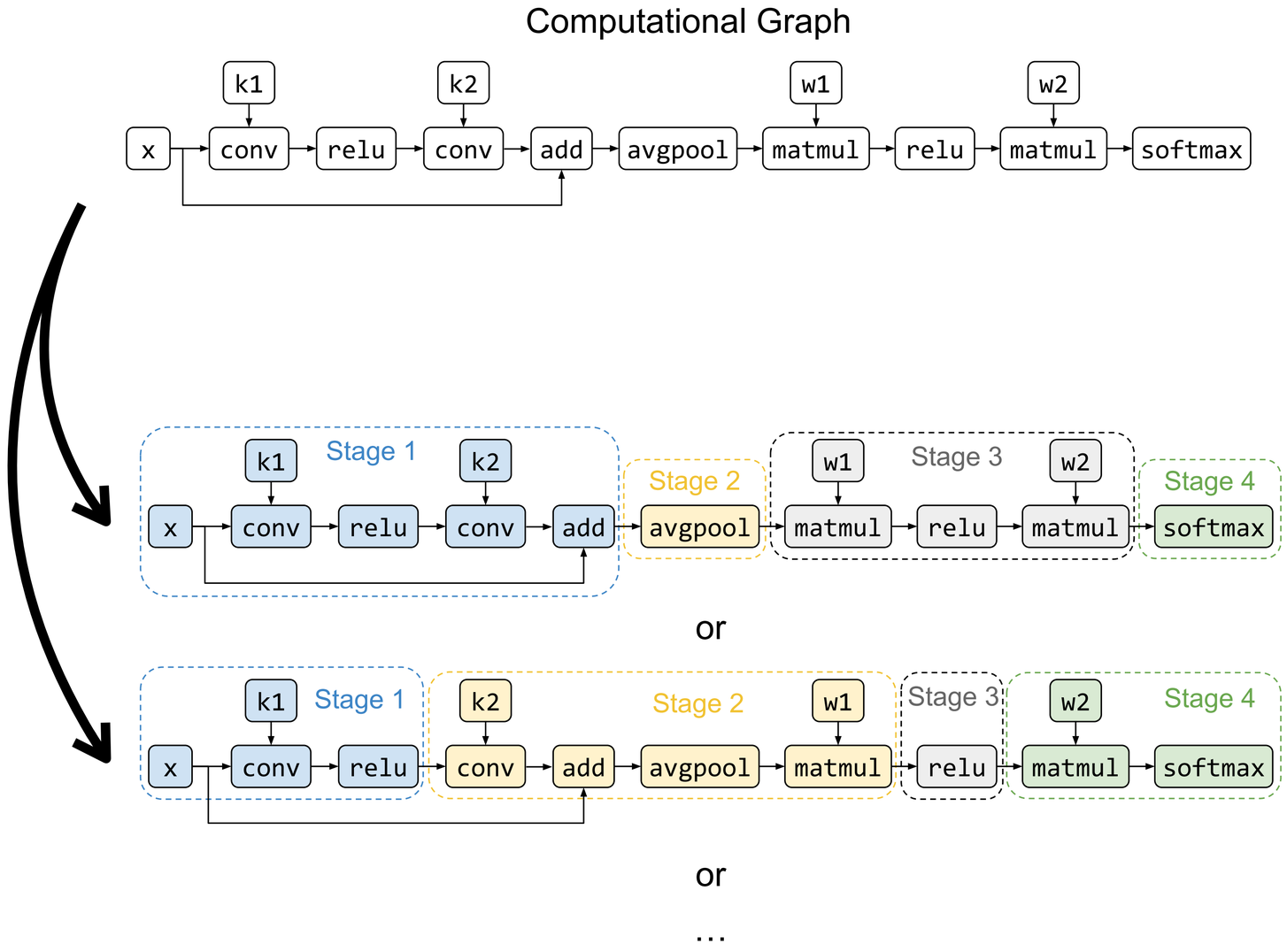
Alpa: A Unified Compiler for Distributed Training

Search for Inter-op Parallelism



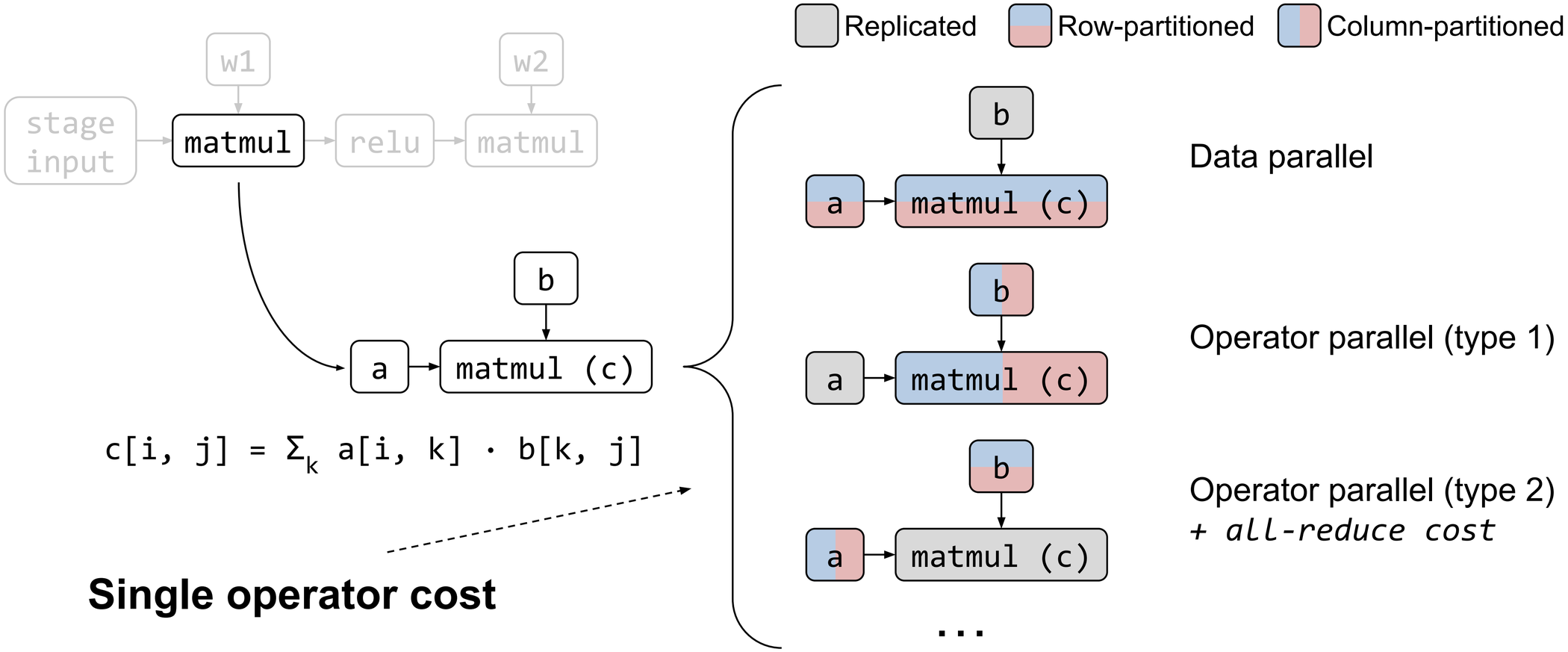
Alpa: A Unified Compiler for Distributed Training

Search for Inter-op Parallelism

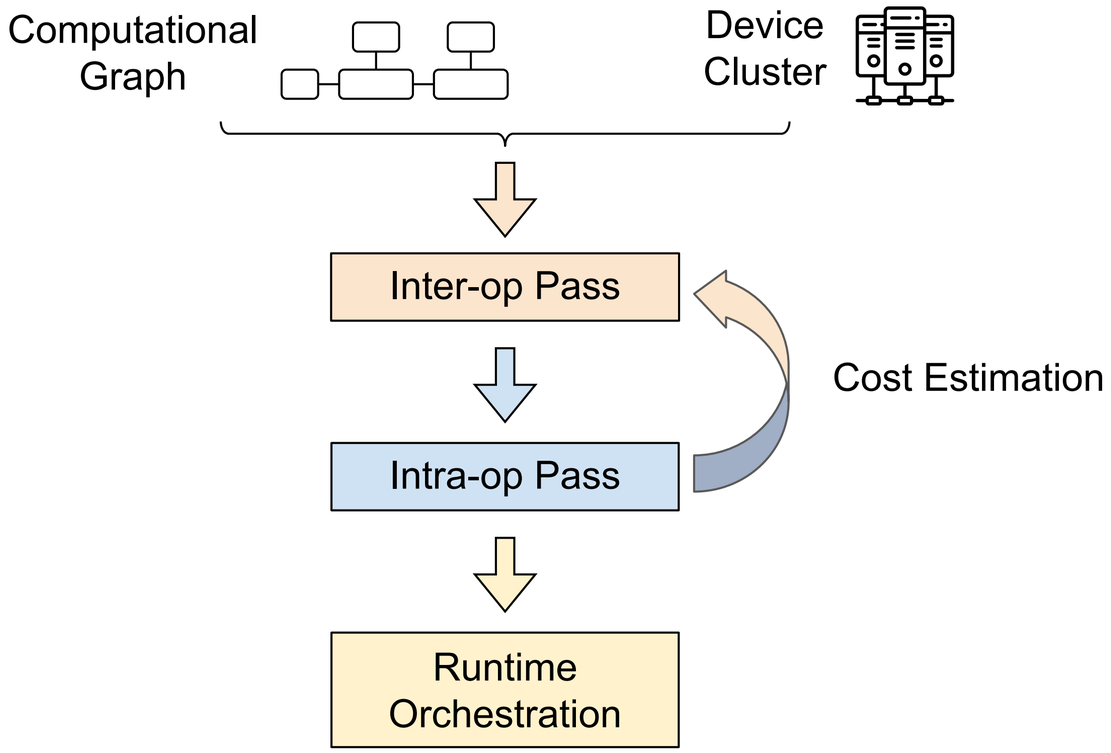


Alpa: A Unified Compiler for Distributed Training

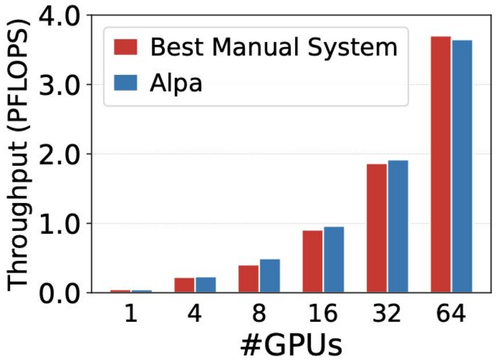
Search for Intra-op Parallelism



Alpa: A Unified Compiler for Distributed Training

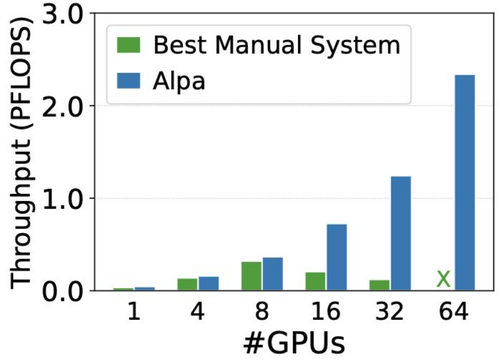


GPT (up to 39B)



Match specialized manual systems.

GShard MoE (up to 70B)



Outperform the manual baseline by up to 8x.

Optimize via Automatic Search, No need to manually config

Alpa-served Meta's OPT-175B: https://opt.alpa.ai
(Large Language Model free to play with)

Summary of Today's Lecture

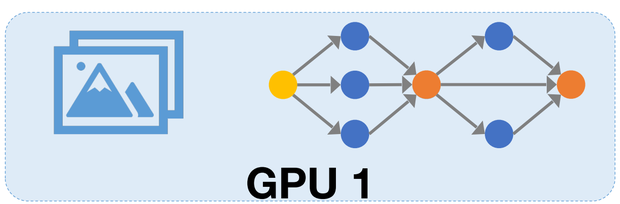
- We learned the basic concepts of distributed training.

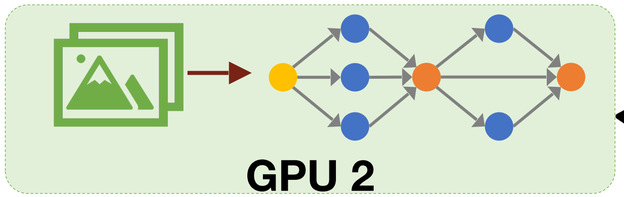
Models	#Params (M)	Training Time (GPU Hours)
ResNet-50	26	31
ResNet-101	45	44
BERT-Base	108	84
Turing-NLG 17B	17,000	TBA
GPT-3 175B	175,000	3,100,000 (**335 years**)

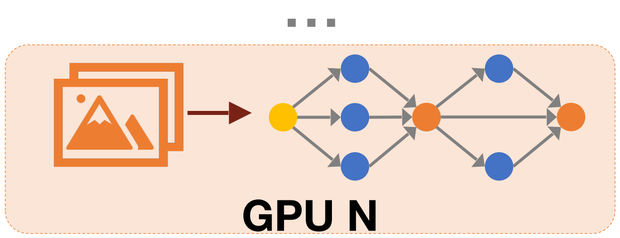
Why distributed training? Accelerate the training process.

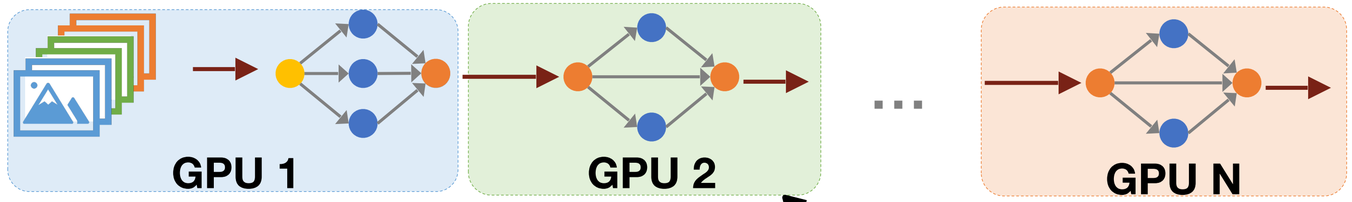
Summary of Today's Lecture

- We learned the basic concepts of distributed training.
- The difference between model parallelism and data parallelism.









Data Parallelism:

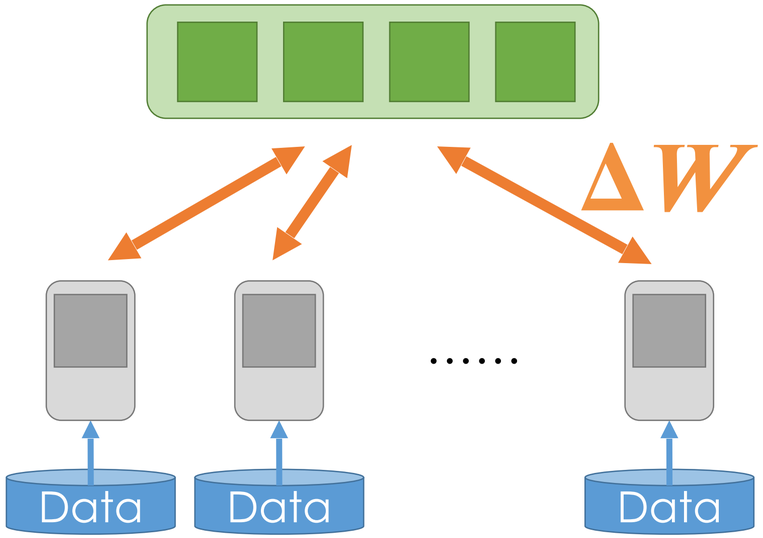
- Split the data

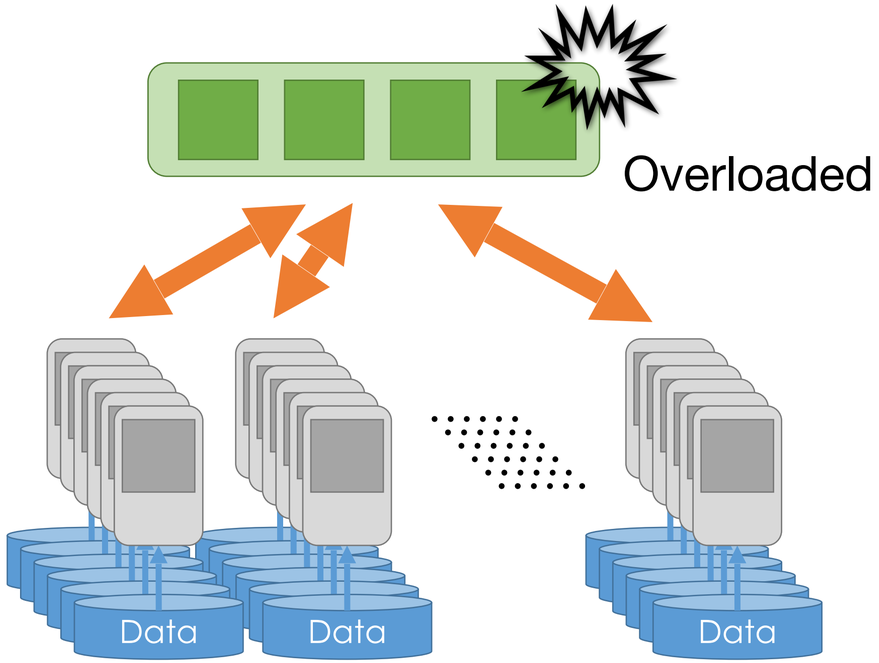
Model Parallelism:

- Split the model

Summary of Today's Lecture

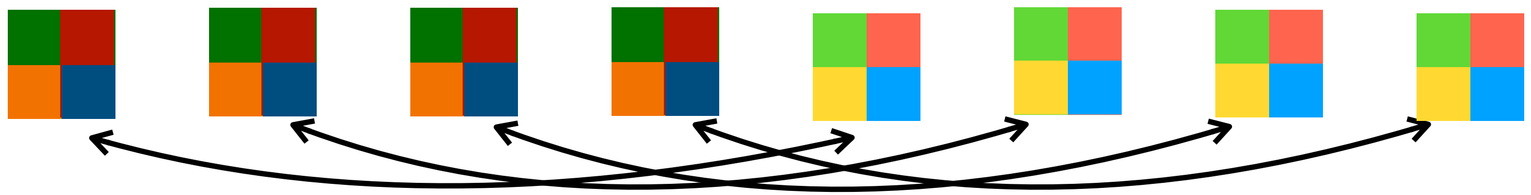
- We learned the basic concepts of distributed training.
- The difference between model parallelism and data parallelism.
- How to scale single node training to multi nodes training with data parallelism.
- The bottleneck of scaling a distributed system is **usually the communication**.





Summary of Today's Lecture

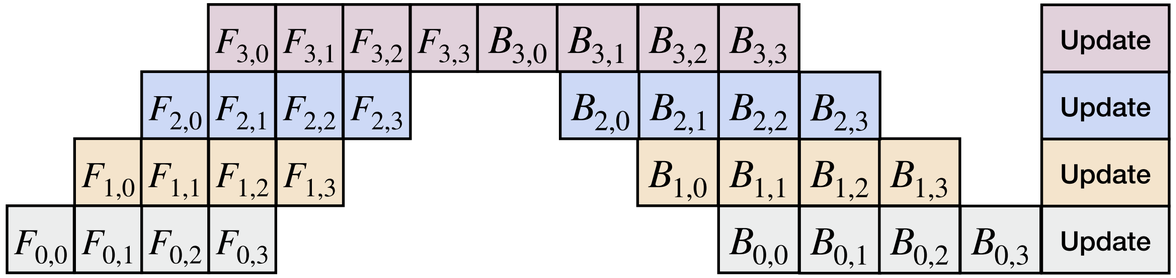
- We learned the basic concepts of distributed training.
- The difference between model parallelism and data parallelism.
- How to scale single node training to multi nodes training with data parallelism.
- The bottleneck of scaling a distributed system is usually the communication.
- The problem of parameter server and how to perform distributed training without parameter server



Recursive Halving AllReduce

Summary of Today's Lecture

- We learned the basic concepts of distributed training.
- The difference between model parallelism and data parallelism.
- How to scale single node training to multi nodes training with data parallelism.
- The bottleneck of scaling a distributed system is usually the communication.
- The problem of parameter server and how to perform distributed training without parameter server
- The workflow of model parallelism and how to improve its utilization.



References

- TSM: Temporal Shift Module for Efficient Video Understanding [Lin et al. 2019]
- Scaling Distributed Machine Learning with the Parameter Server. [Mu Li et al. 2014]
- Optimization of collective communication operations in [Rajeev et al. 2005]
- GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
- Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training [Li et al. 2021]
- Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning [Zheng et al. 2022]