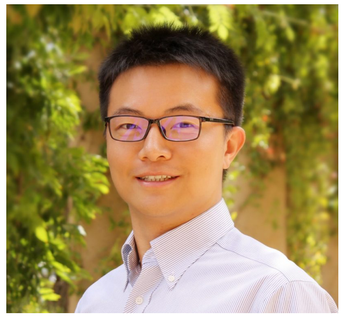
MIT HAN LAB

EfficientML.ai Lecture 17 Distributed Training

Part I



Song Han

Associate Professor, MIT
Distinguished Scientist, NVIDIA

@SongHan_MIT

Lecture Plan

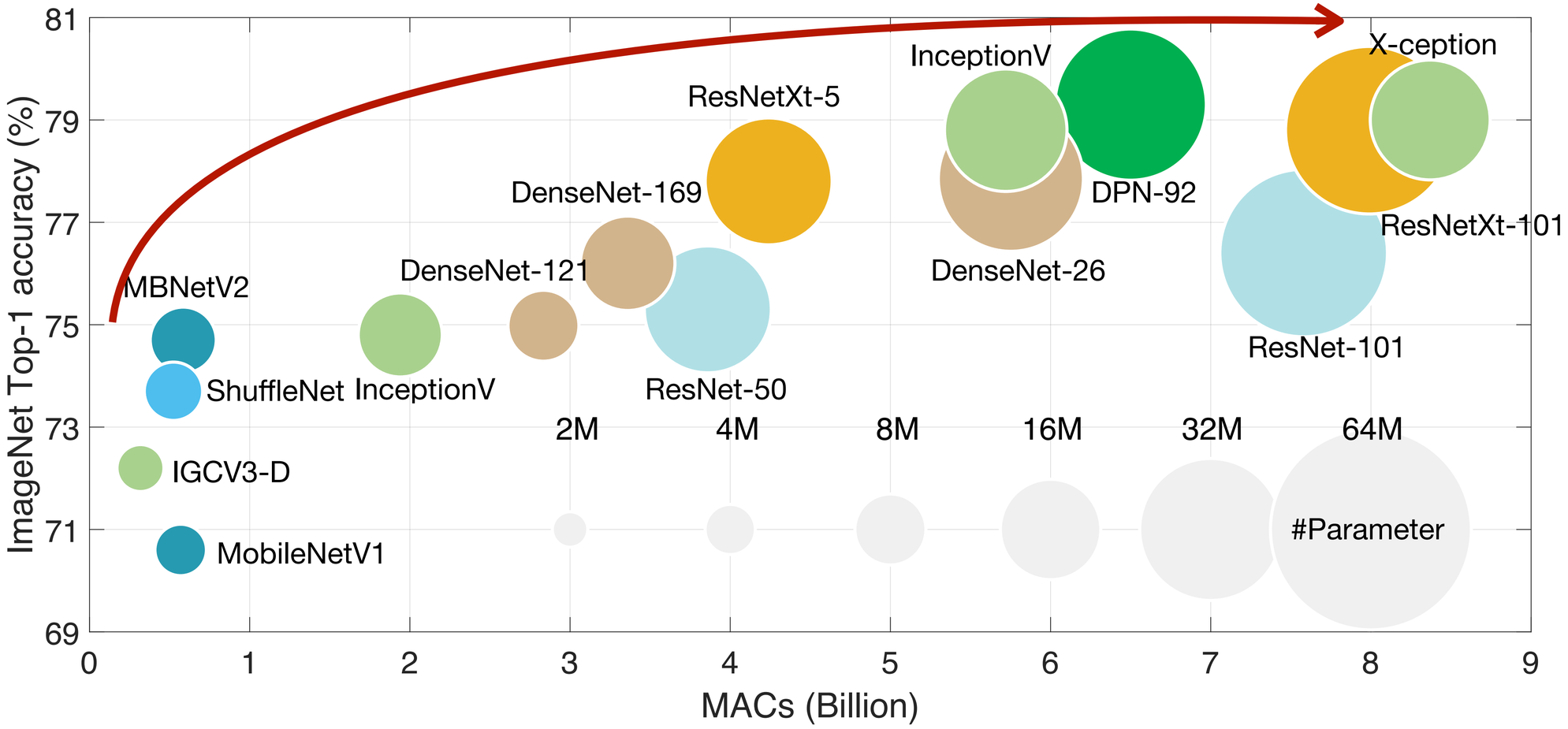
1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

Lecture Plan

1. **Background and motivation**
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

Models are Getting Larger

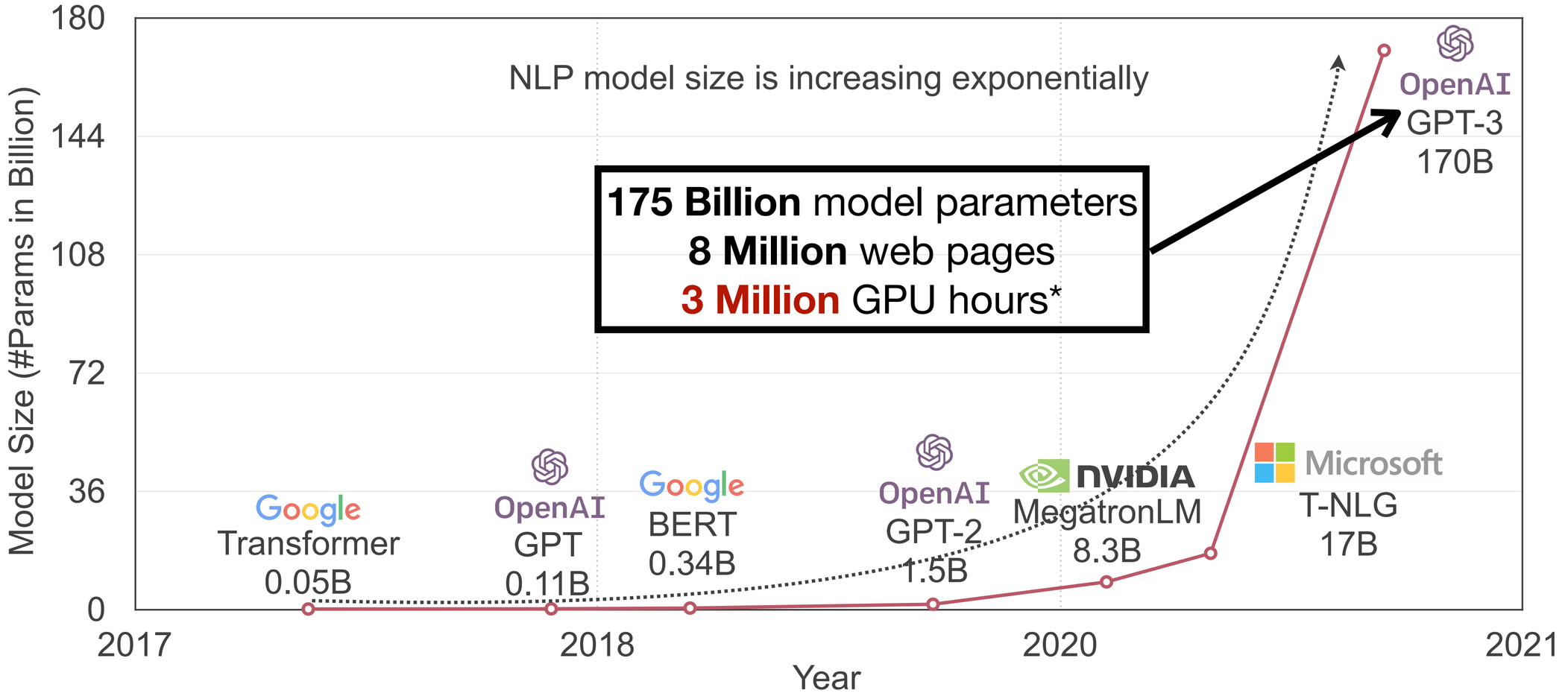
Better model always comes with higher computational cost (vision)



Figures from Once-for-all project page.

Models are Getting Larger

Better model always comes with higher computational cost (NLP)



Figures style adapted from Microsoft Turing Project

* Numbers referenced from https://lambdalabs.com/blog/demystifying-gpt-3

Models are Getting 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 **355 years** to finish GPT-3!

* Numbers referenced from https://lambdalabs.com/blog/demystifying-gpt-3

Models are Getting 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 Example

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
- Storage: 250 PB
- Connection: Dual-rail EDR InfiniBand network of 200 Gbit/s

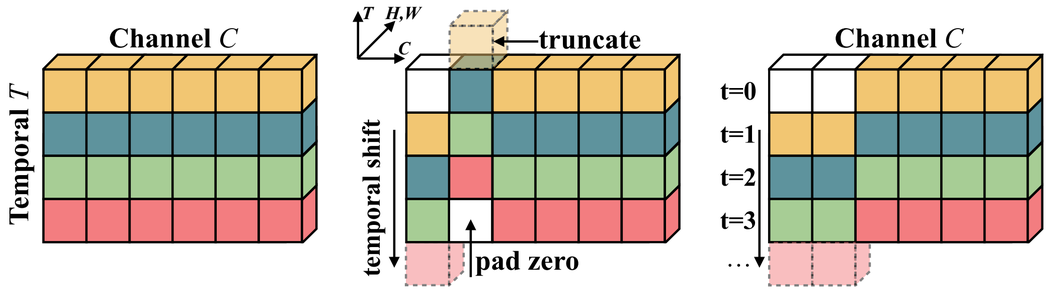
Large-Scale Distributed Training Example

Train a Video Model (TSM) 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]

Lecture Plan

Understand how distributed training works and improve the efficiency

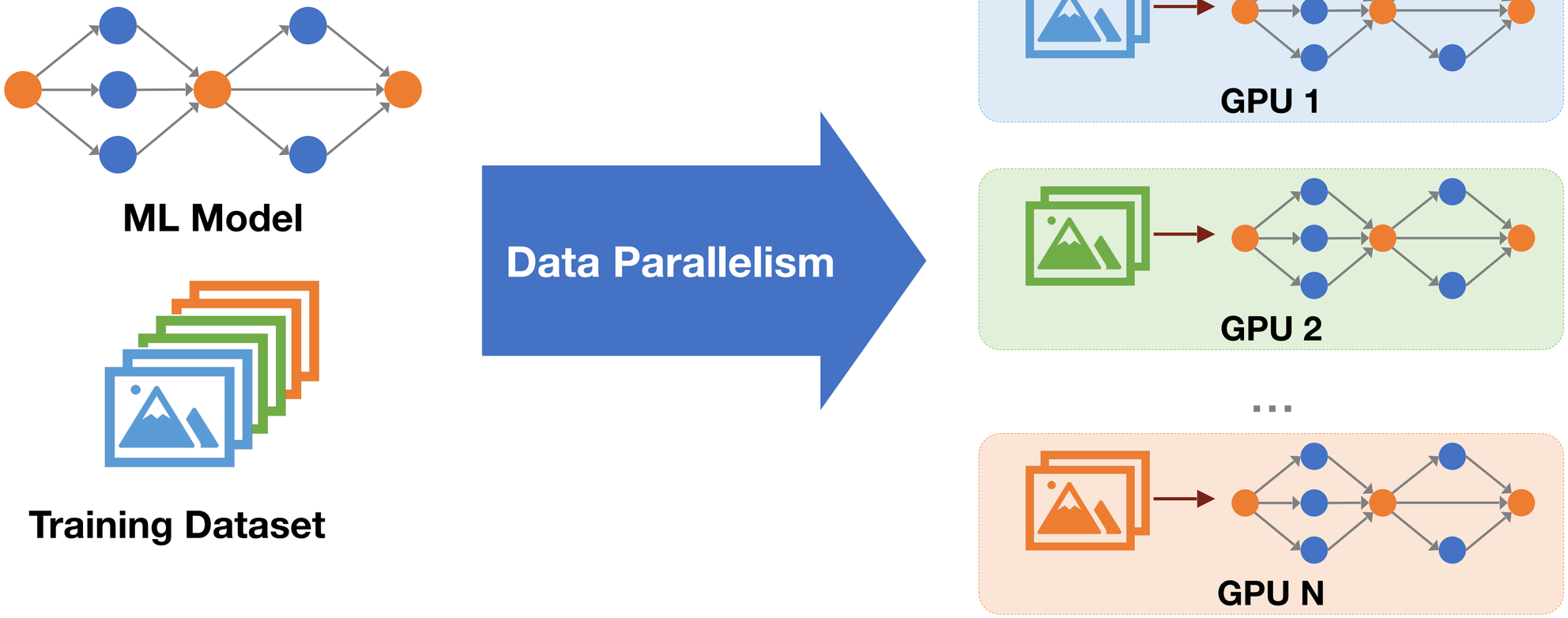
1. Background and motivation
2. **Parallelization methods for distributed training**
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

Parallelization Methods in Distributed Training

- **Data Parallelism**
- Pipeline Parallelism
- Tensor Parallelism

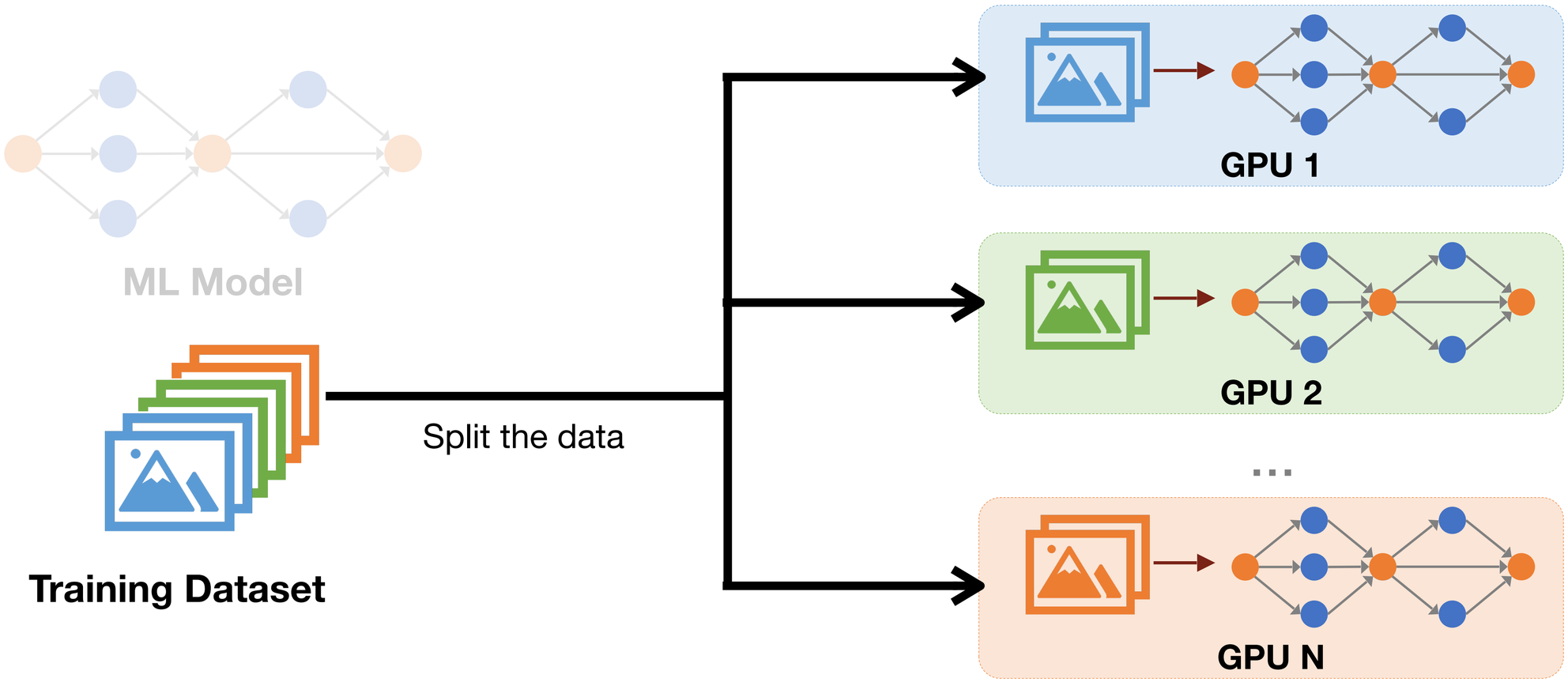
Parallelization Methods in Distributed Training

Data Parallelism



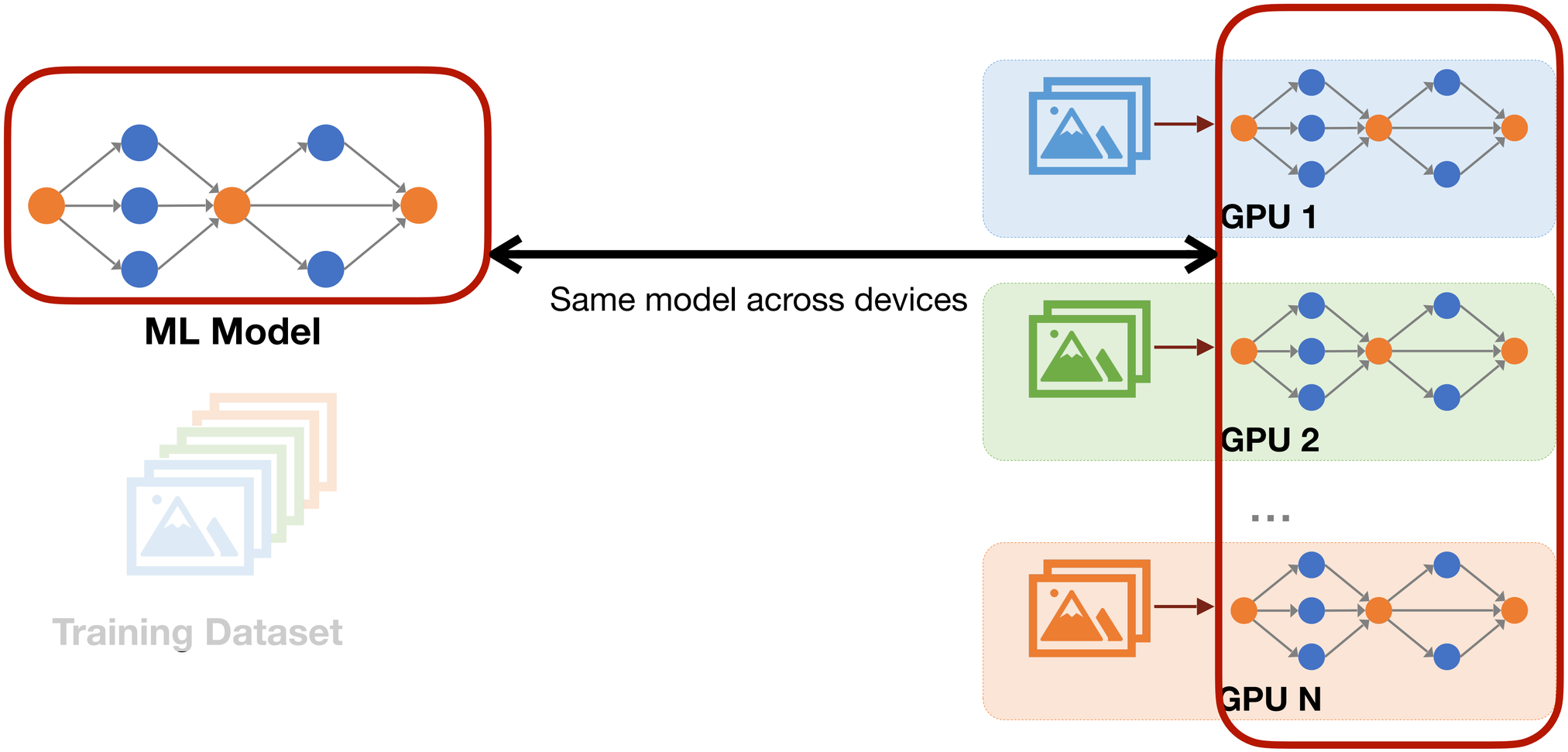
Parallelization Methods in Distributed Training

Data Parallelism



Parallelization Methods in Distributed Training

Data Parallelism



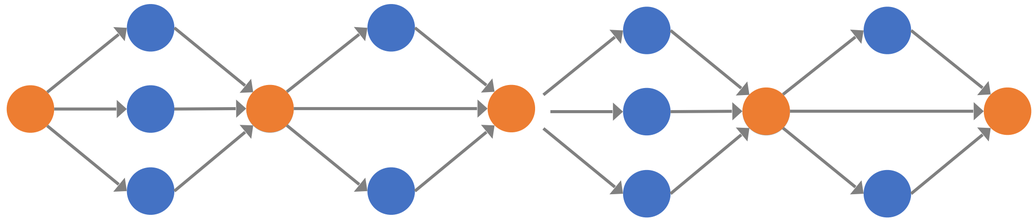
Parallelization Methods in Distributed Training

- Data Parallelism
- **Pipeline Parallelism**
- Tensor Parallelism

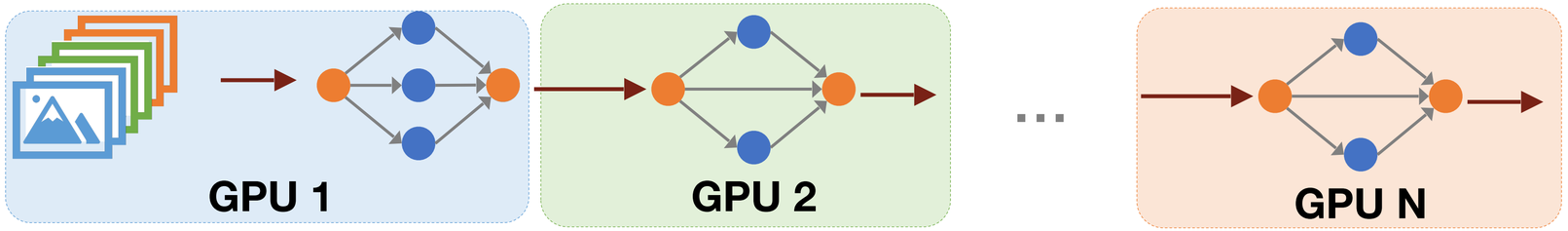
Parallelization Methods in Distributed Training

Pipeline Parallelism

Training Dataset



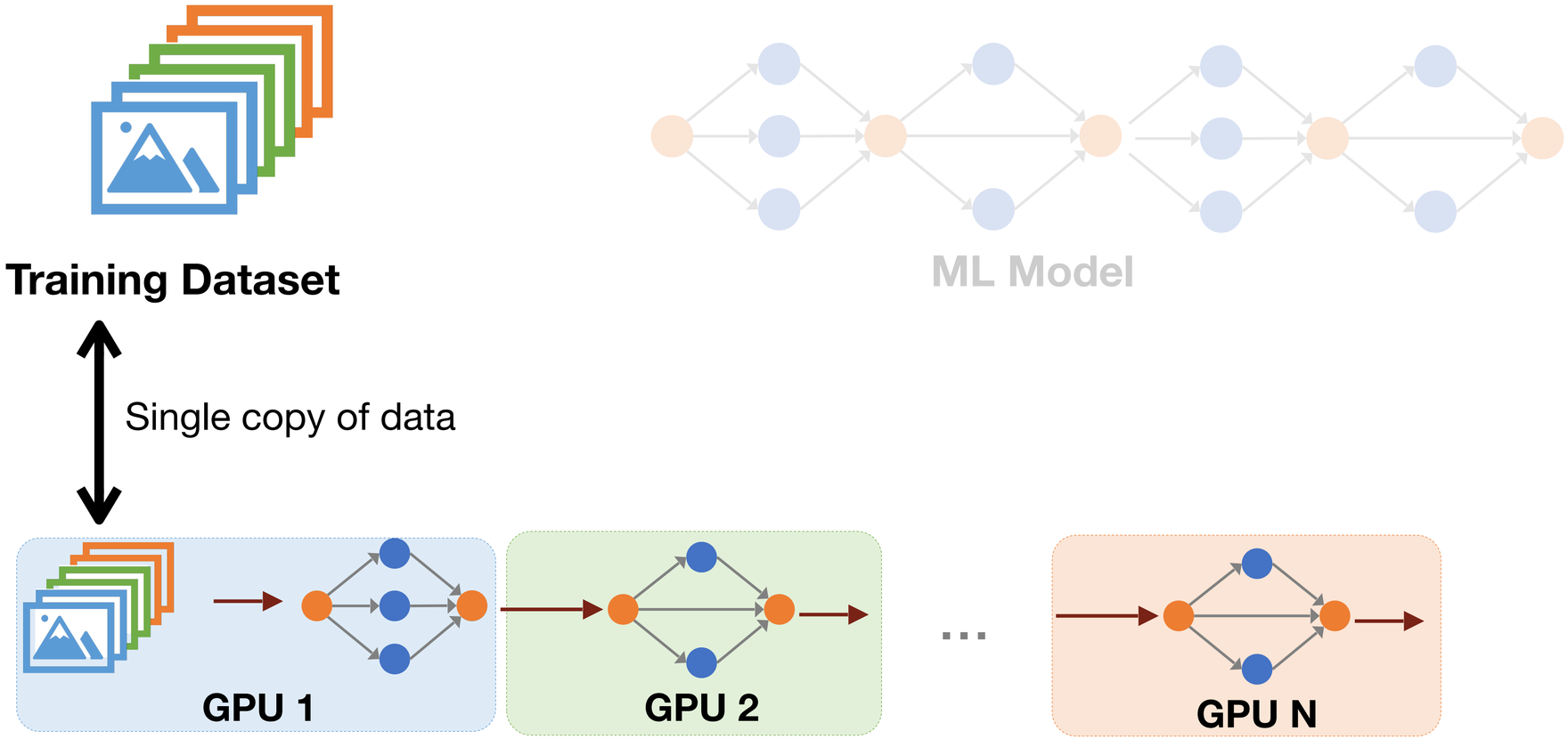
ML Model



Figures credit from CMU 15-849 [Jia 2022]

Parallelization Methods in Distributed Training

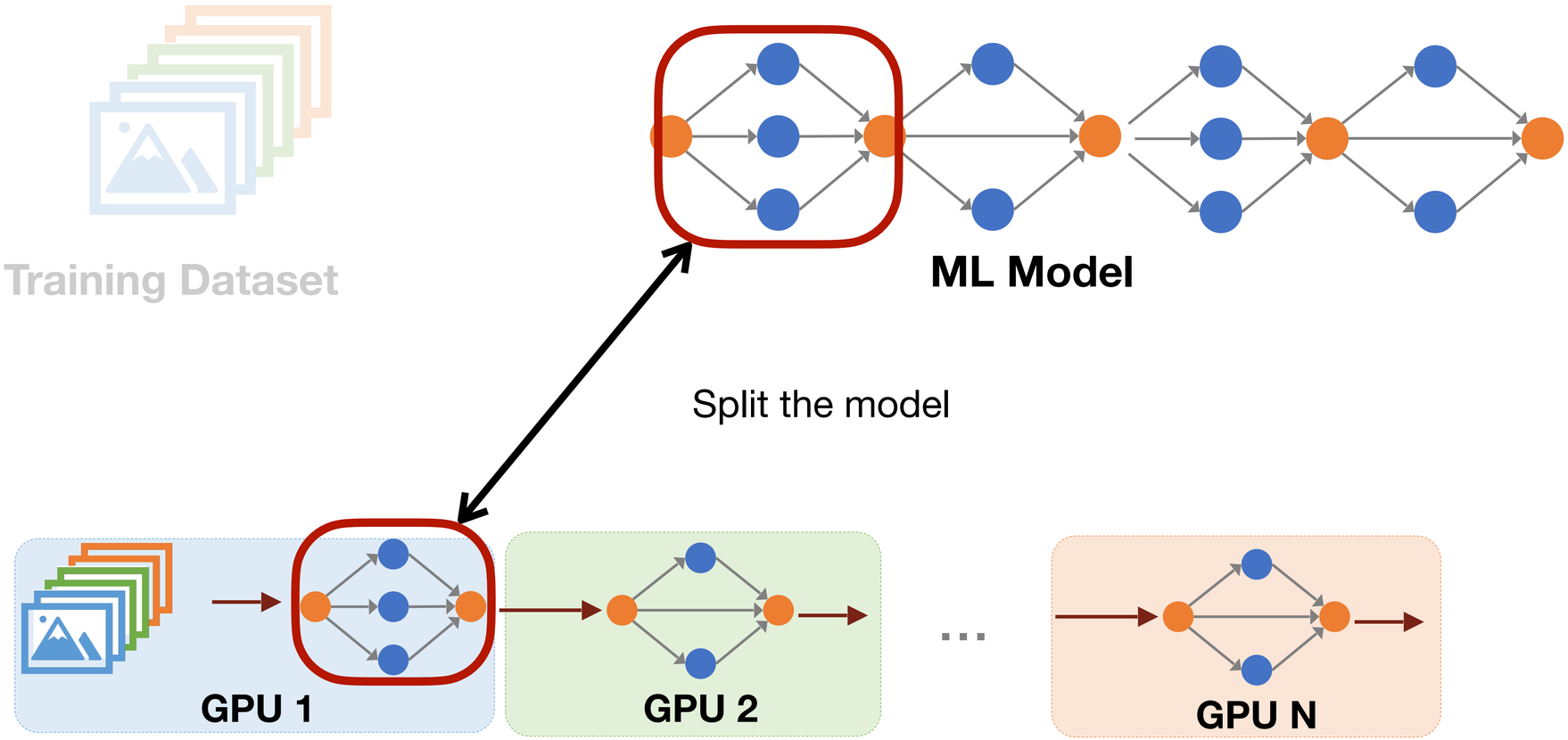
Pipeline Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Parallelization Methods in Distributed Training

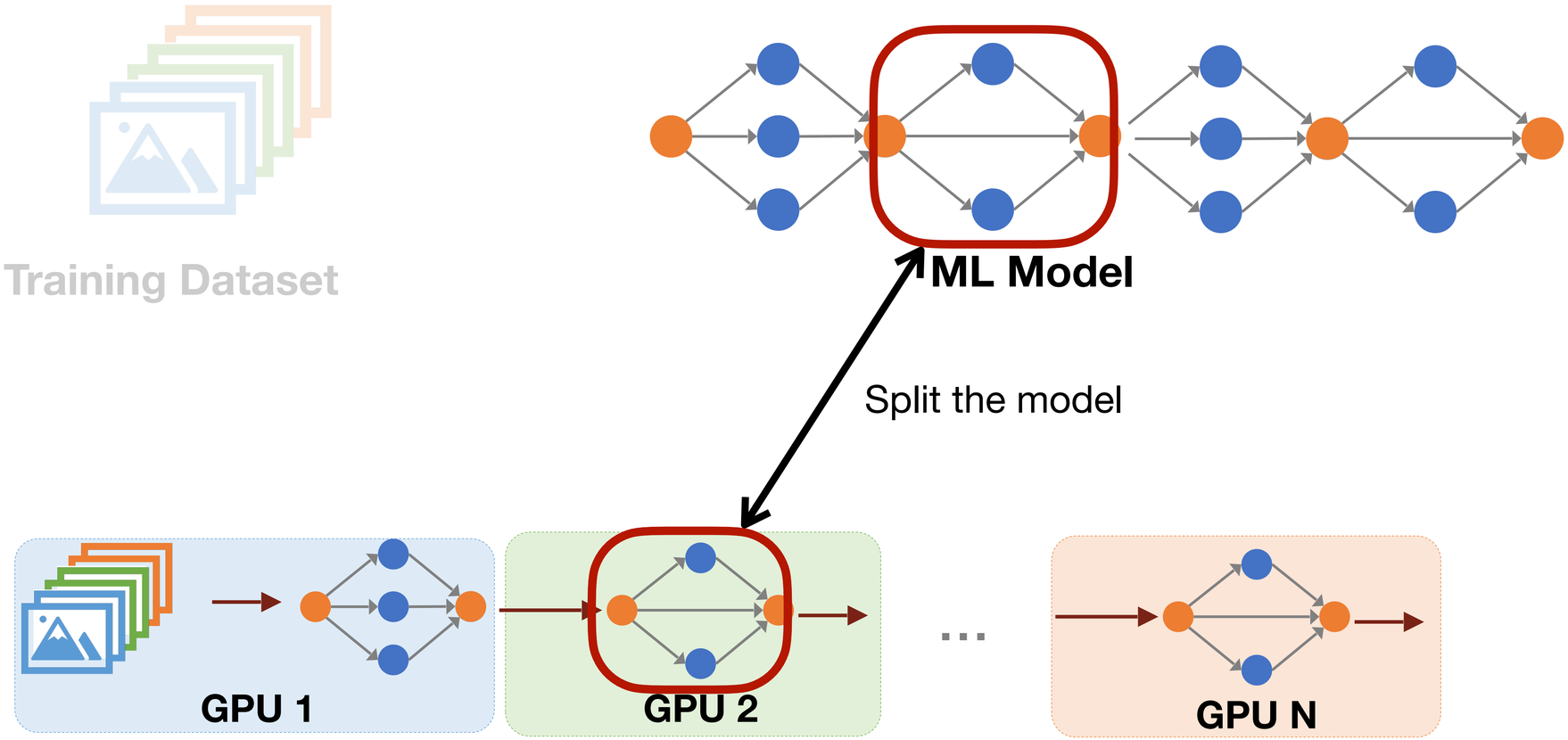
Pipeline Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Parallelization Methods in Distributed Training

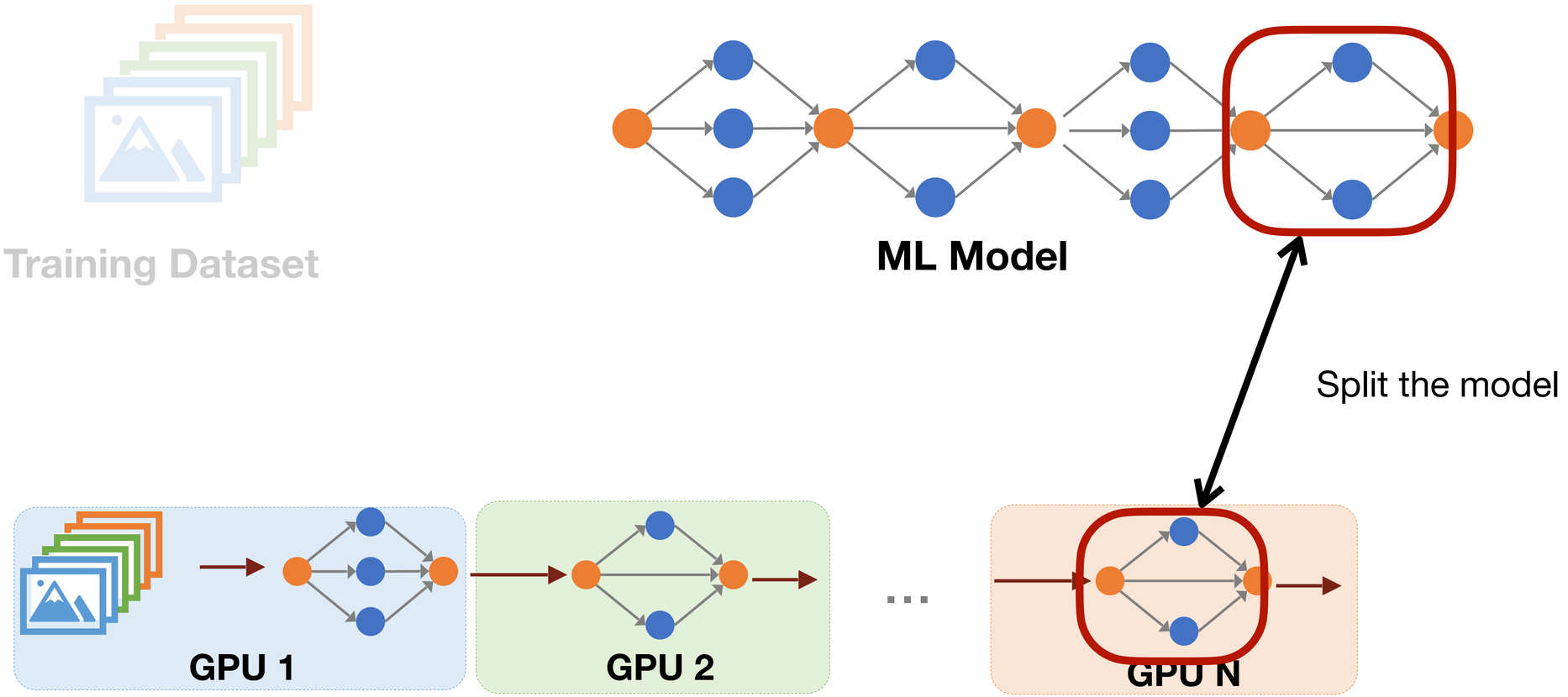
Pipeline Parallelism



Figures credit from CMU 15-849 [Jia 2022]

Parallelization Methods in Distributed Training

Pipeline Parallelism



Figures credit from CMU 15-849 [Jia 2022]

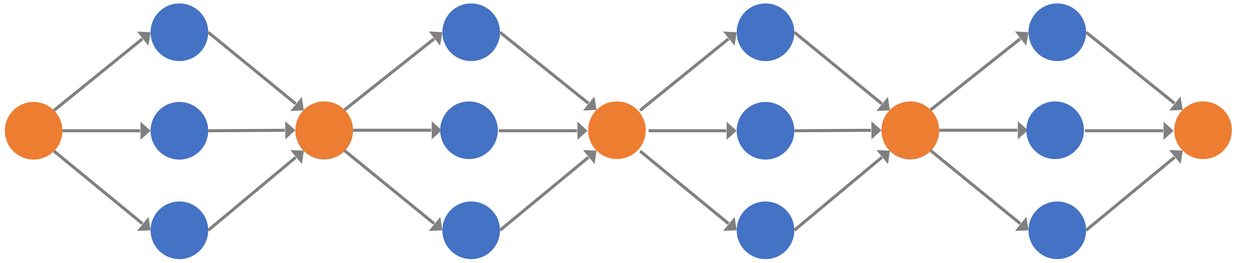
Parallelization Methods in Distributed Training

- Data Parallelism
- Pipeline Parallelism
- **Tensor Parallelism**

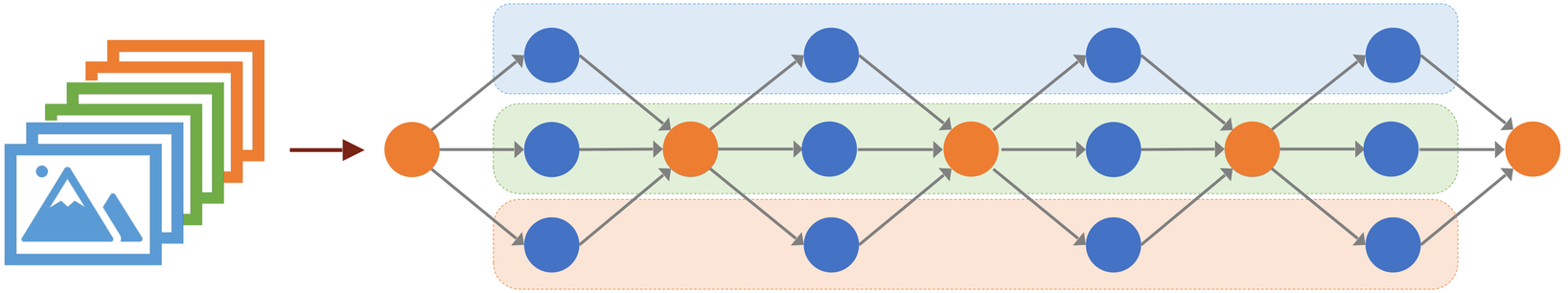
Parallelization Methods in Distributed Training

Tensor Parallelism

Training Dataset



ML Model



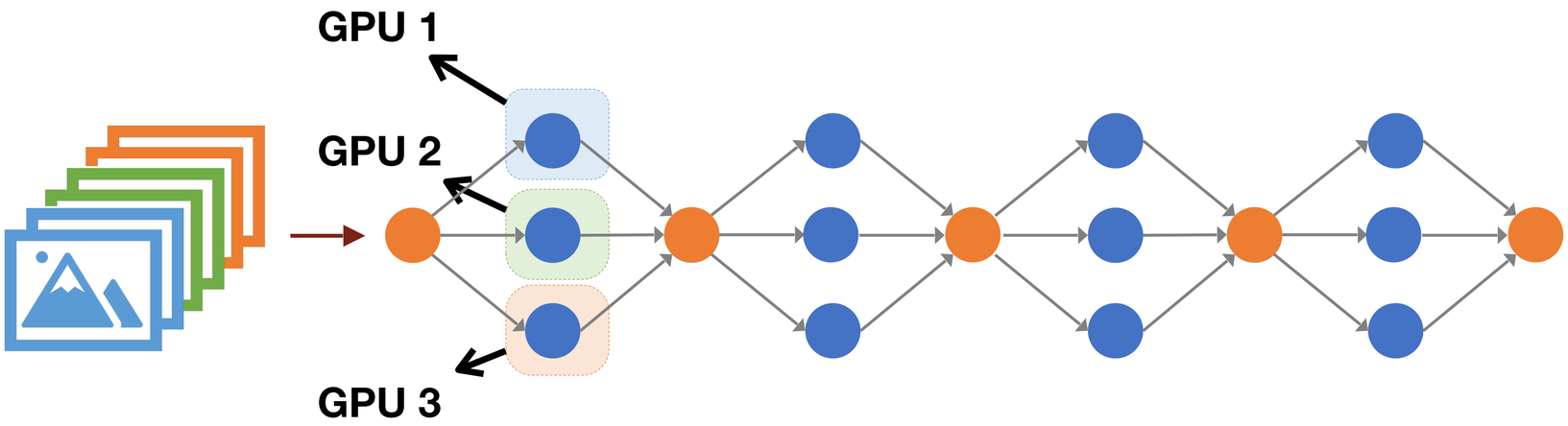
Parallelization Methods in Distributed Training

Tensor Parallelism

Training Dataset

ML Model

Split the activations then sync



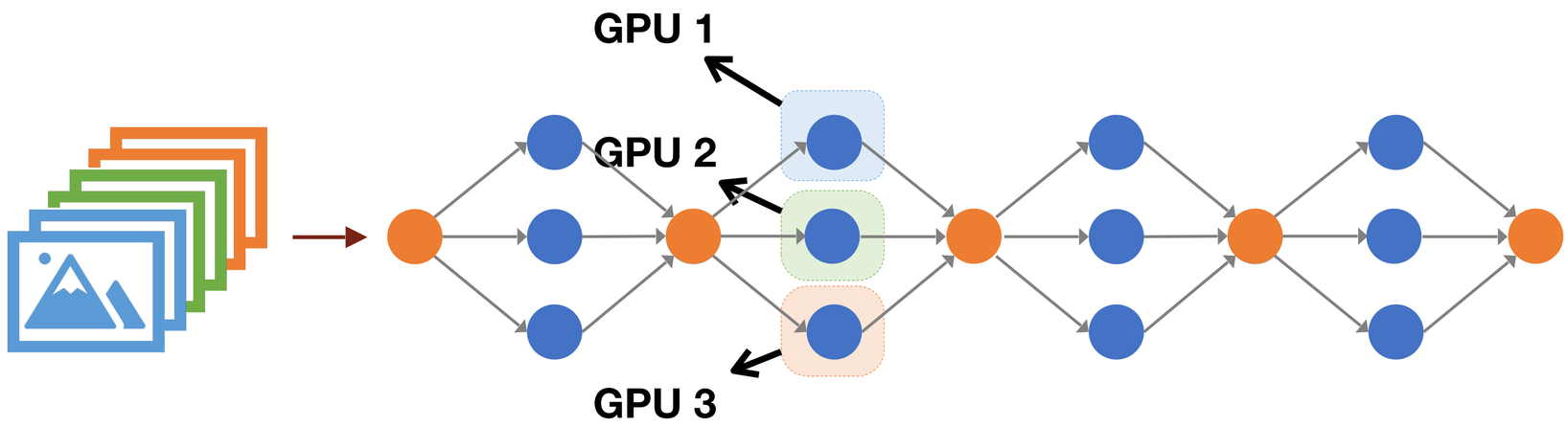
Parallelization Methods in Distributed Training

Tensor Parallelism

Training Dataset

ML Model

Split the activations then sync



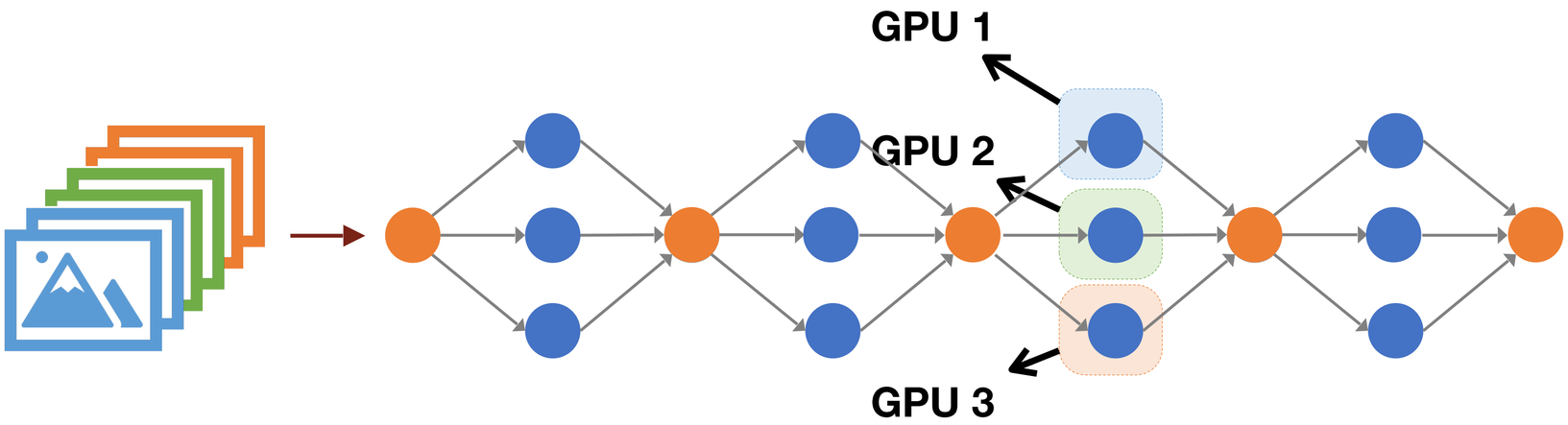
Parallelization Methods in Distributed Training

Tensor Parallelism

Training Dataset

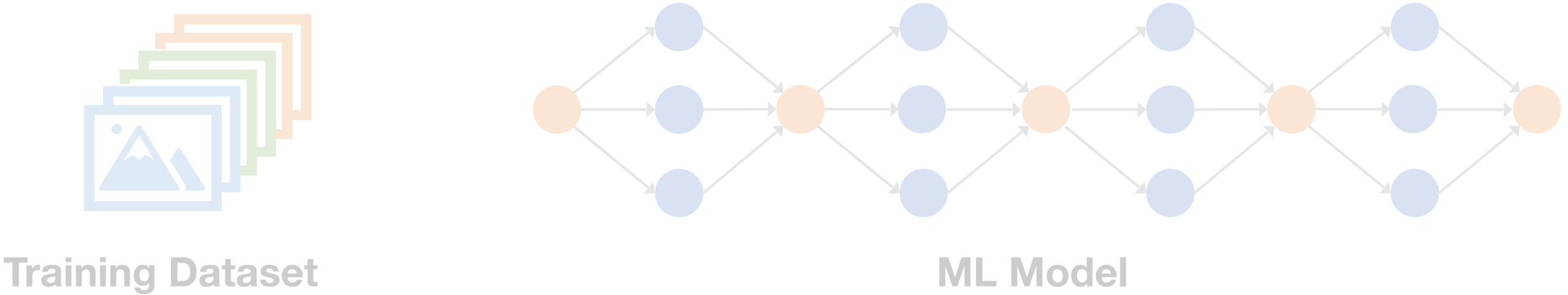
ML Model

Split the activations then sync

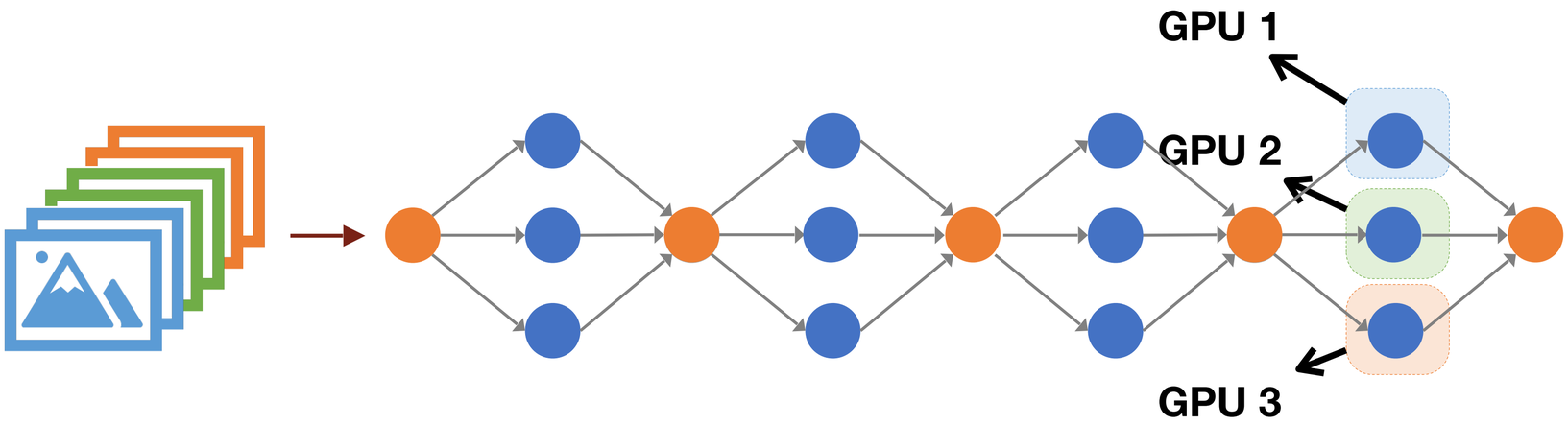


Parallelization Methods in Distributed Training

Tensor Parallelism



Split the activations then sync



Lecture Plan

Understand how distributed training works and improve the efficiency

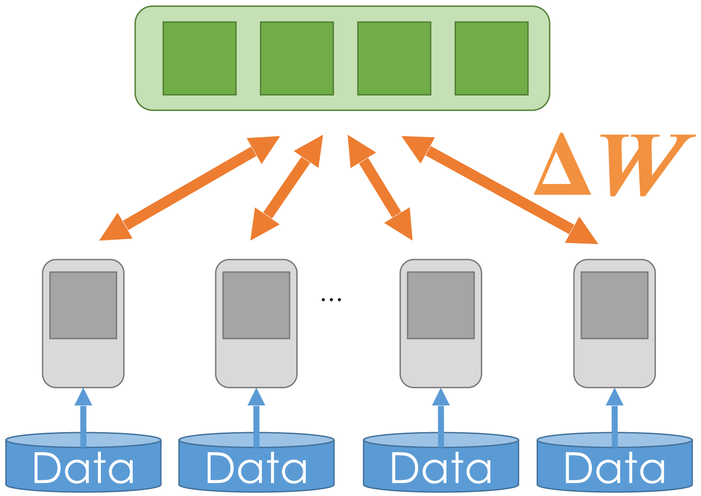
1. Background and motivation
2. Parallelization methods for distributed training
3. **Data parallelism**
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

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.

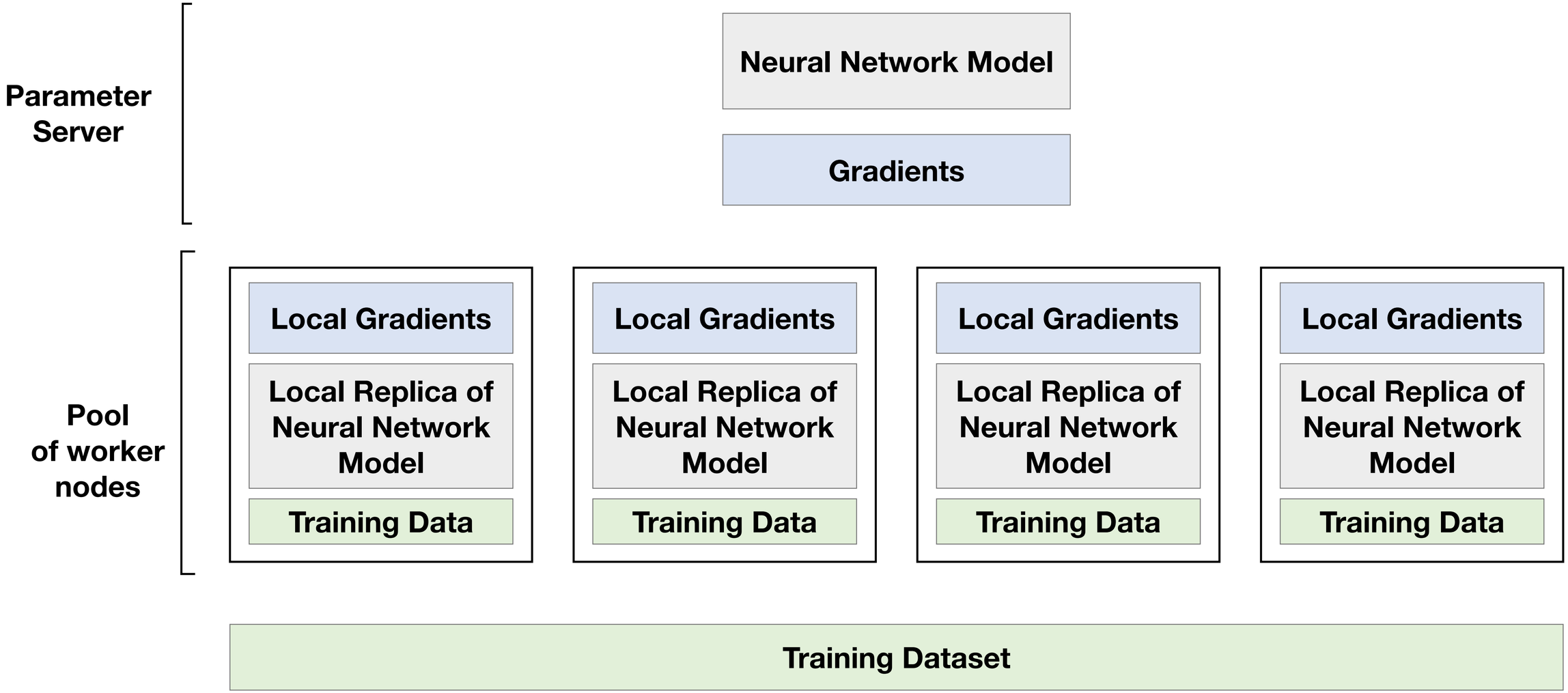
Two different roles in framework:

- 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

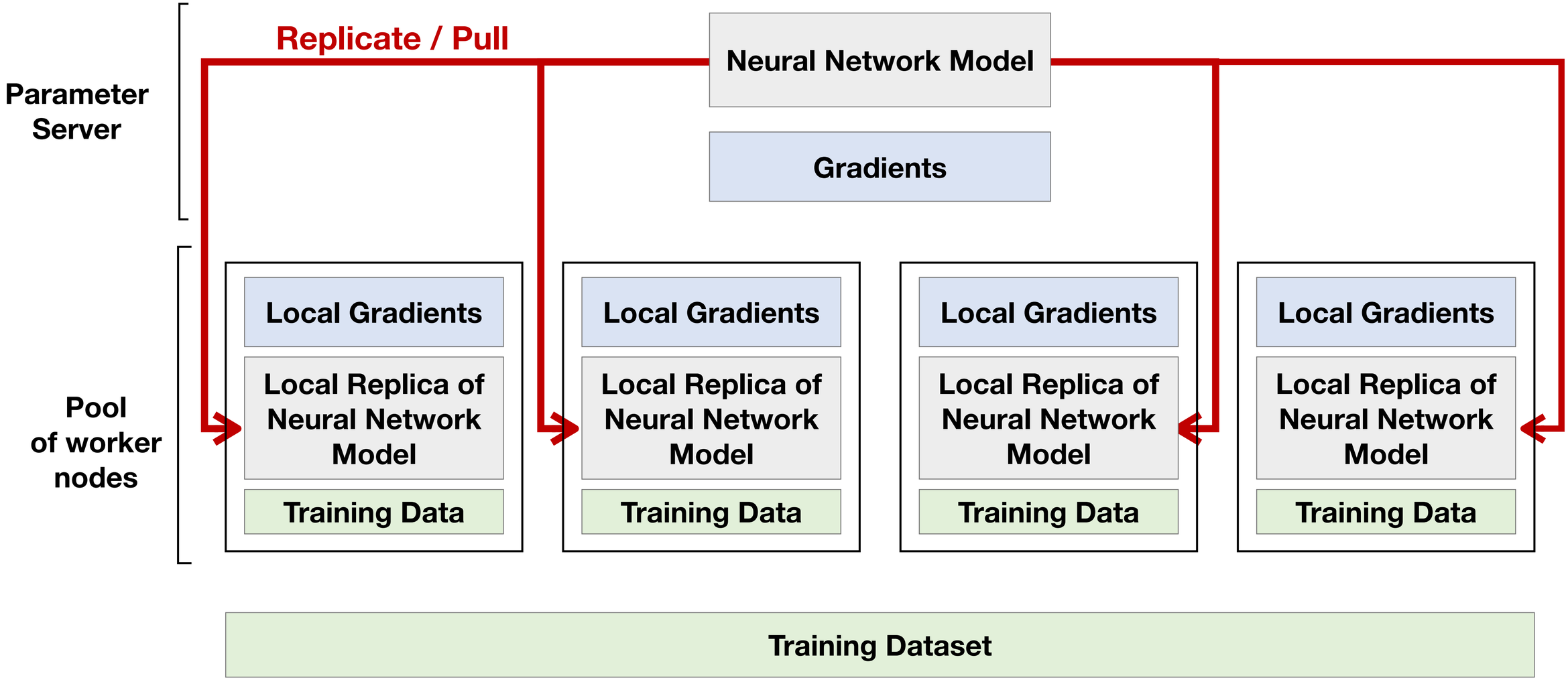
Data Parallelism

Infrastructure Overview



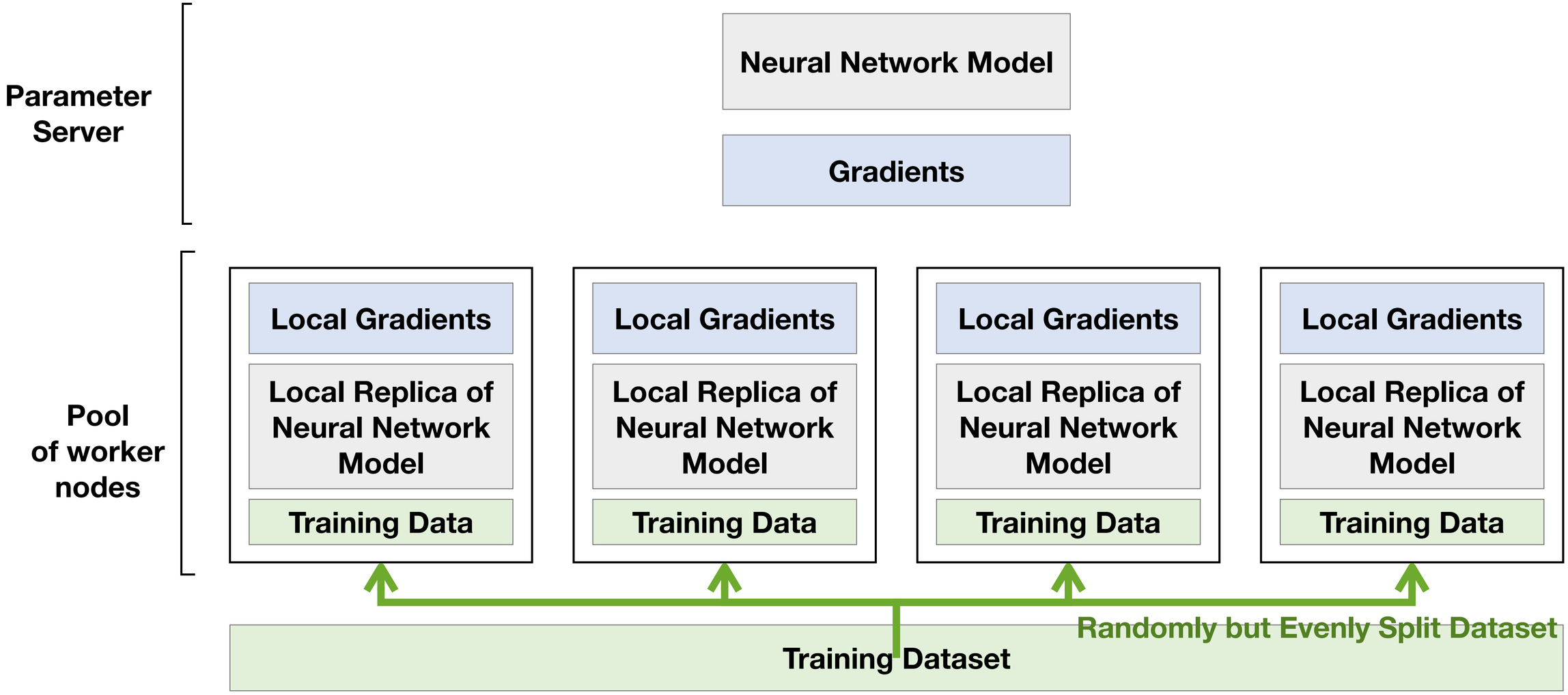
Data Parallelism

1 - Replicate Models to Each Worker



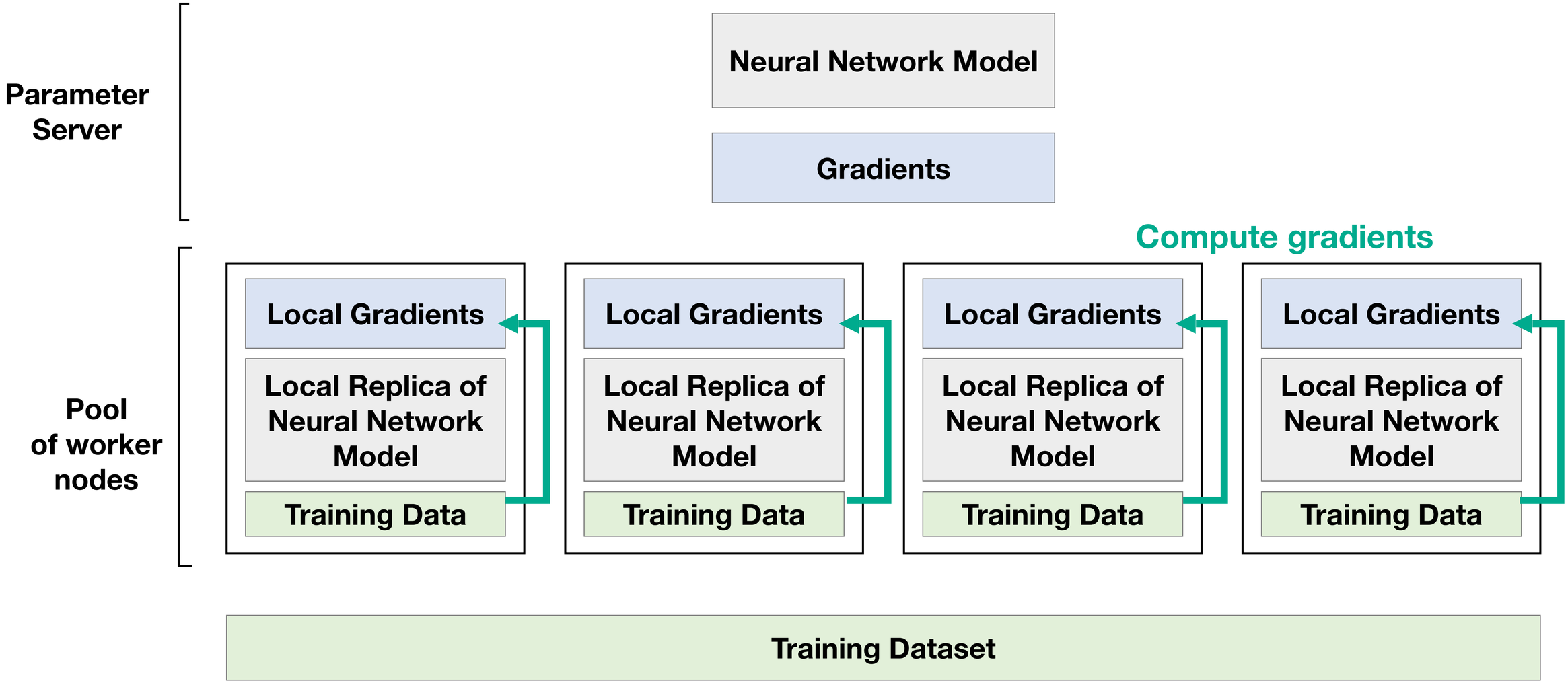
Data Parallelism

2 - Split Training Data to Workers



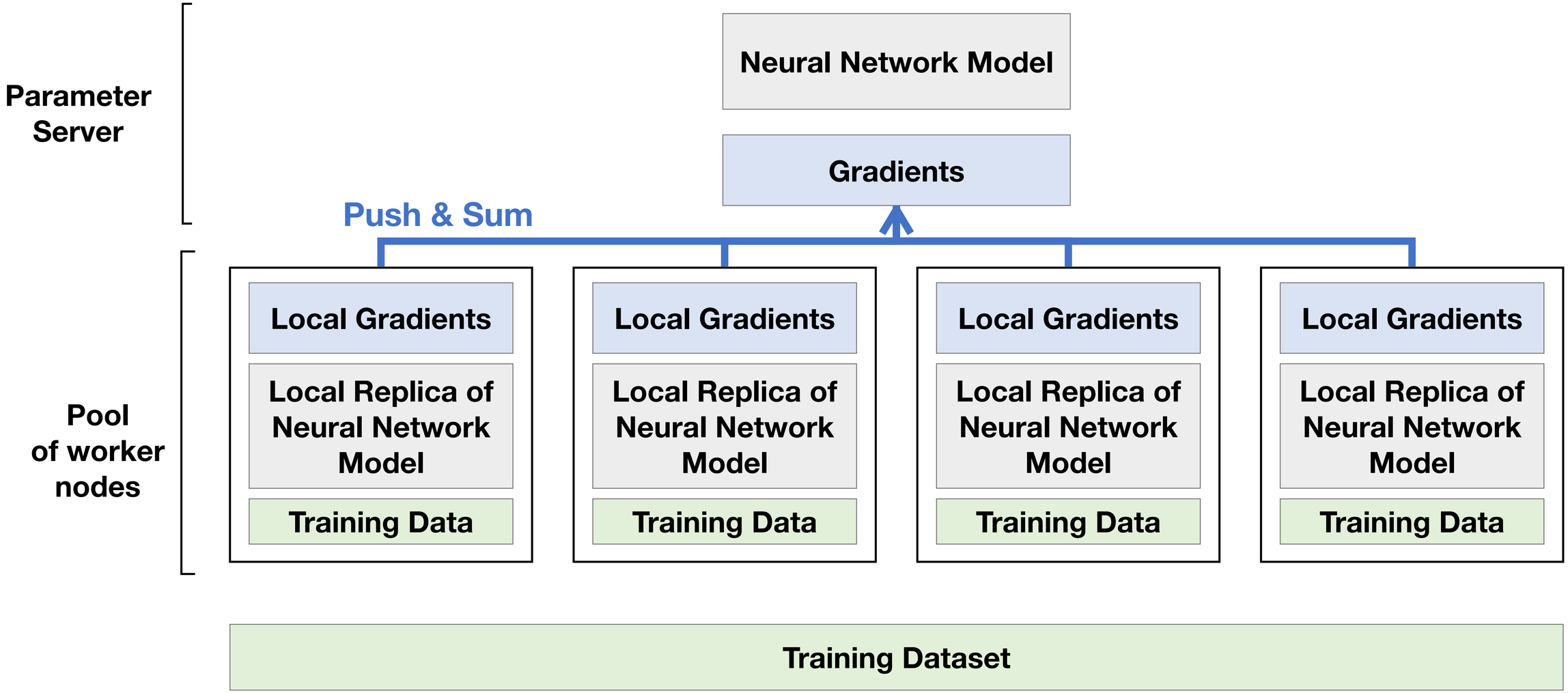
Data Parallelism

3 - Compute gradients



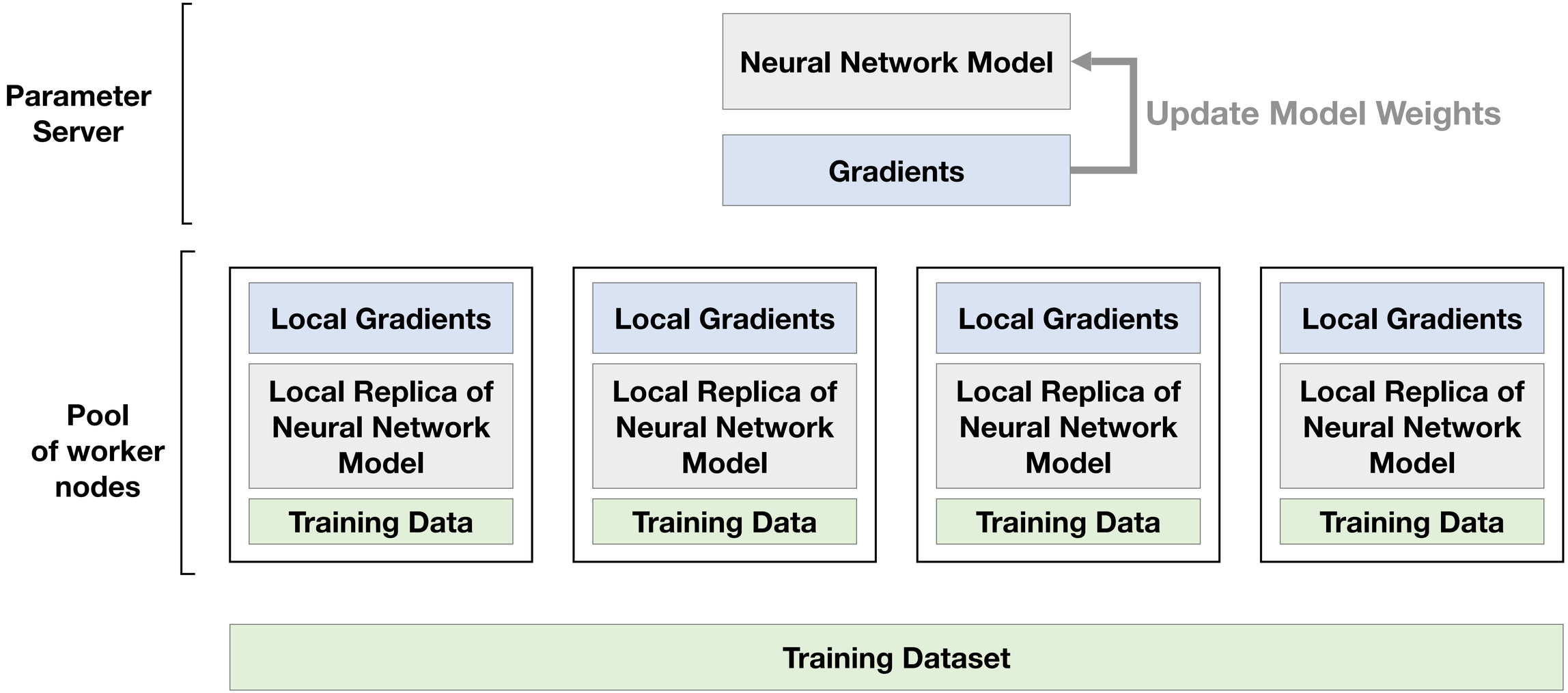
Data Parallelism

4 - Aggregate and Synchronize Gradients



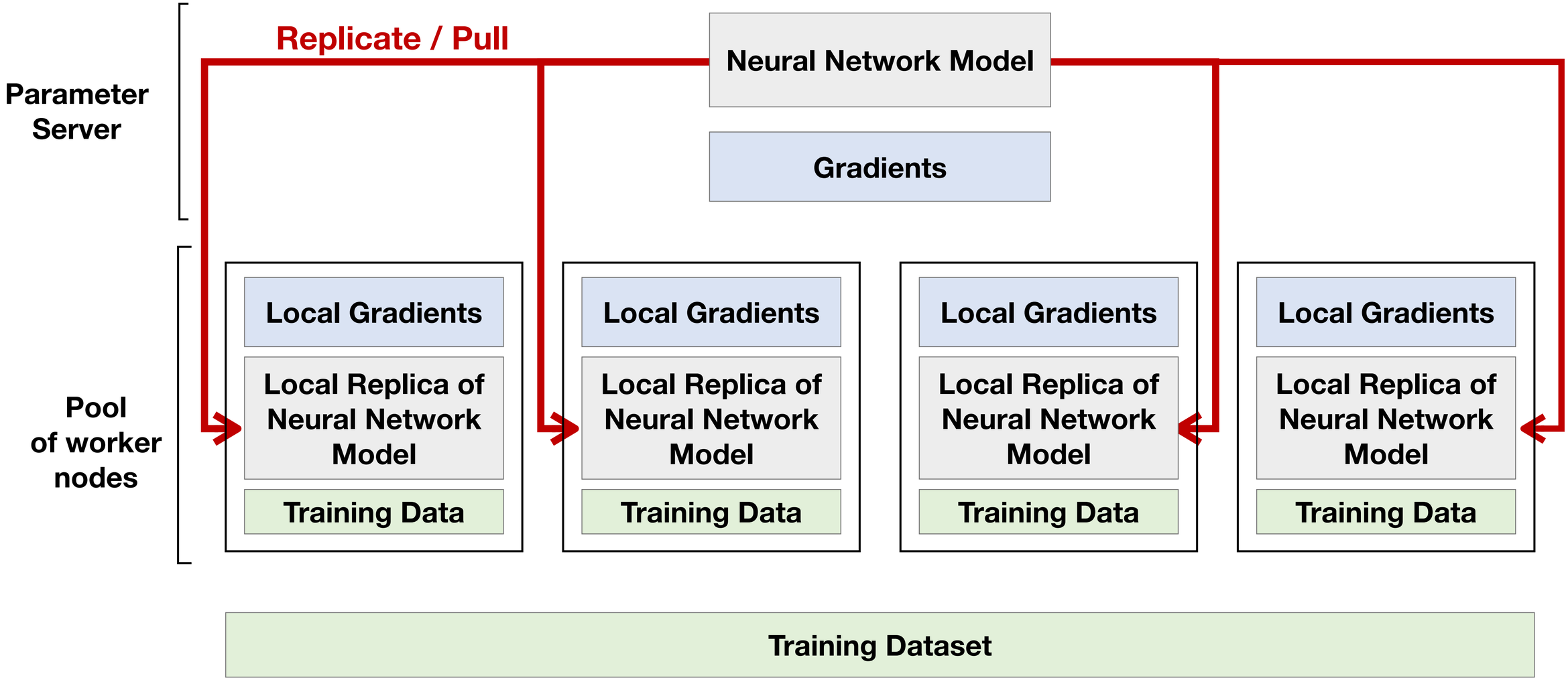
Data Parallelism

5 - Perform Gradient Step and Update Model Parameters



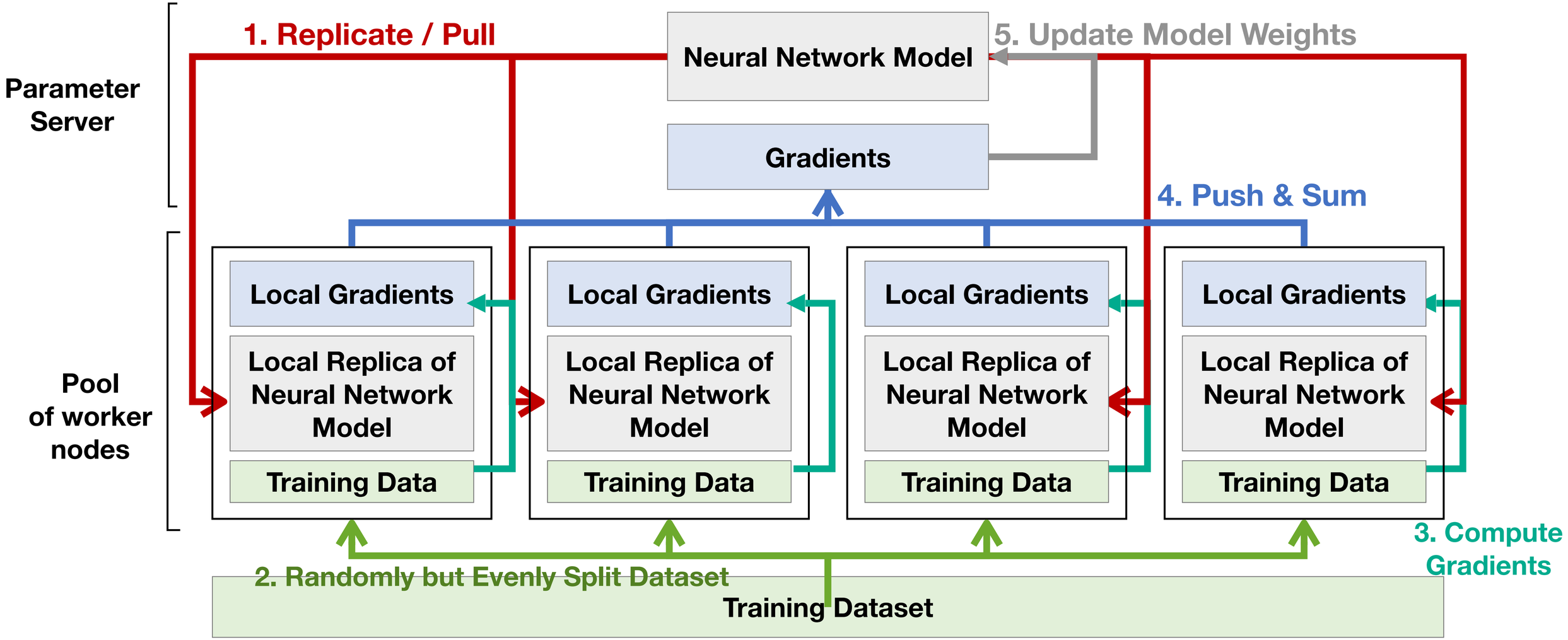
Data Parallelism

Repeats Previous Steps



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.

two synchronization steps

Lecture Plan

Understand how distributed training works and improve the efficiency

1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. **Communication primitives**
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

Communication Primitives

One-to-One: Send and Recv

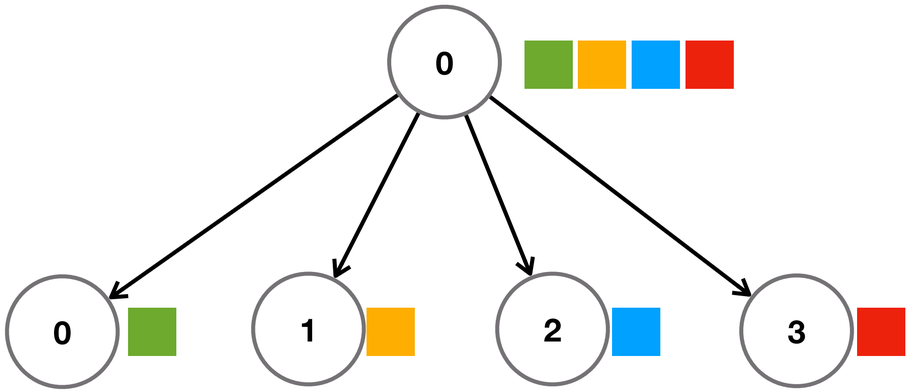
One-to-one communication: transfer data from one process to another

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

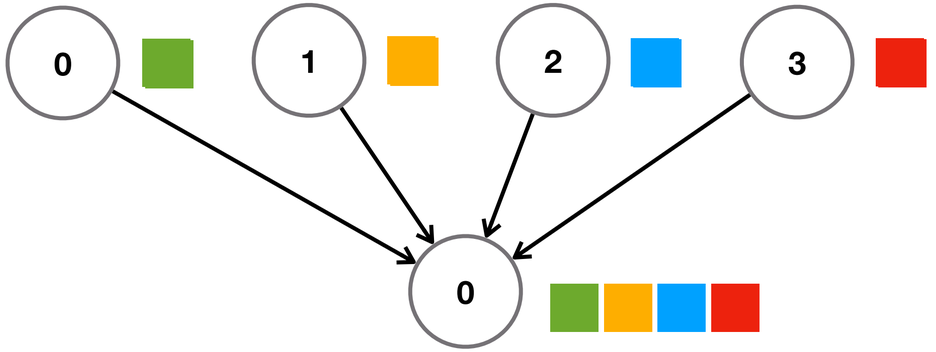
Communication Primitives

One-to-Many: Scatter and Gather

Scatter



Gather

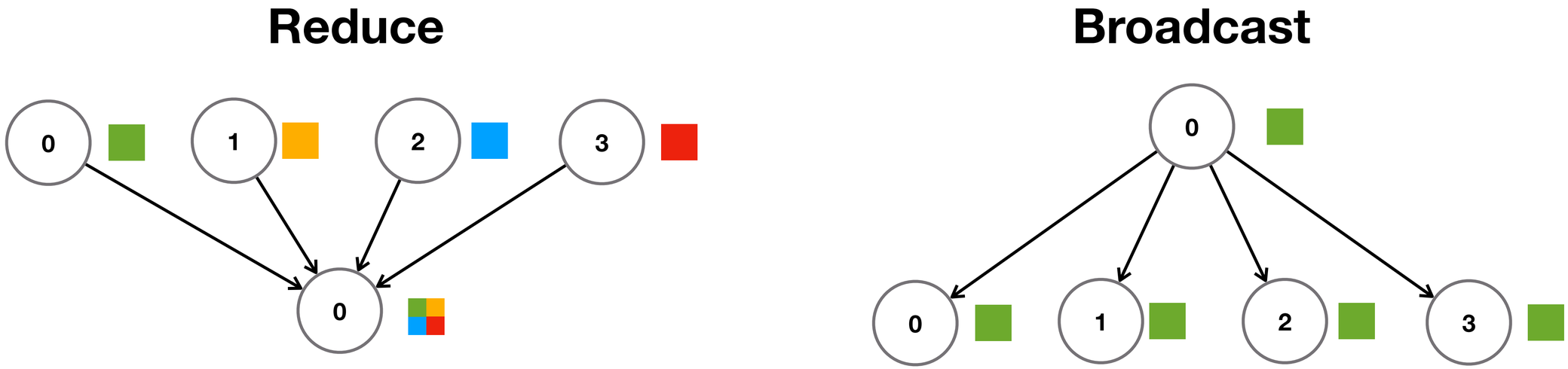


One-to-many communication: a type of operations performed across all workers.

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

Communication Primitives

One-to-Many: Reduce and Broadcast



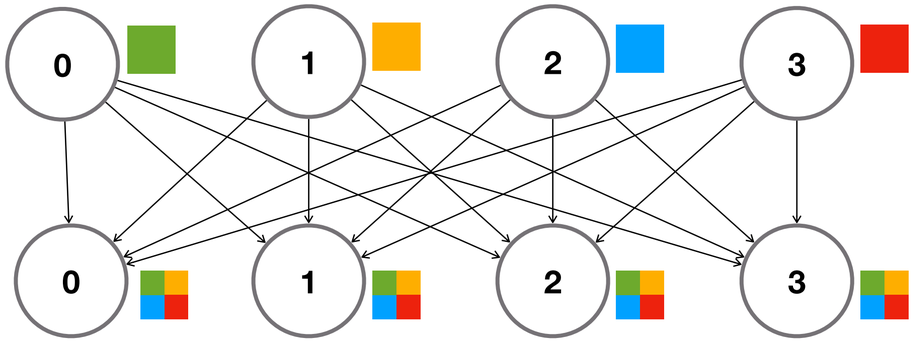
One-to-many communication: a type of operations performed across all workers.

- 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]
- Broadcast: send identical copies to all other workers.

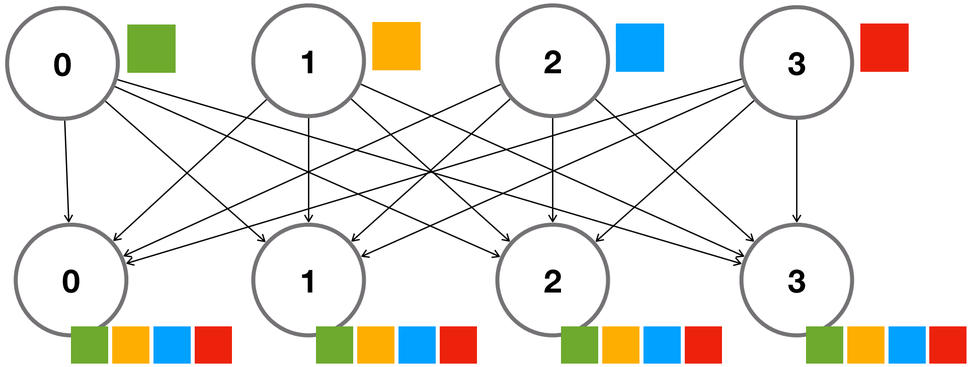
Communication Primitives

Many-to-Many: All Reduce and All Gather

All-Reduce



All-Gather

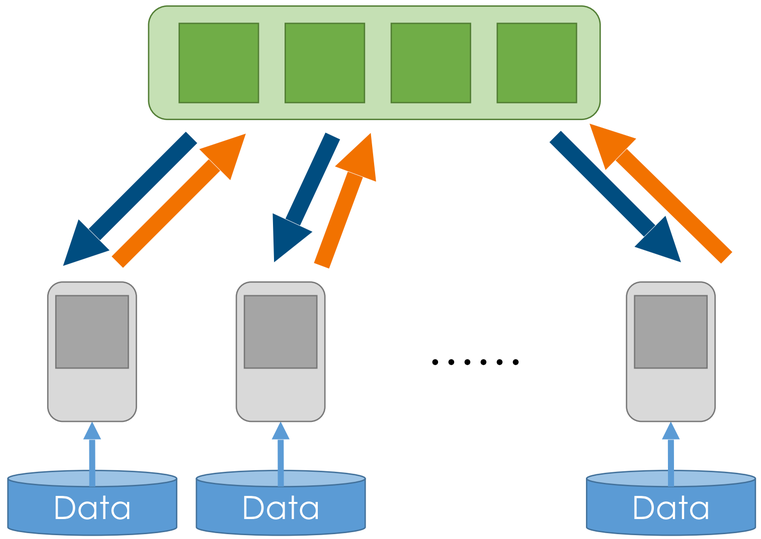


Many-to-many communication:

- All-Reduce: perform *Reduce* on all workers
- All-Gather: perform *Gather* on all workers

Communication Primitives

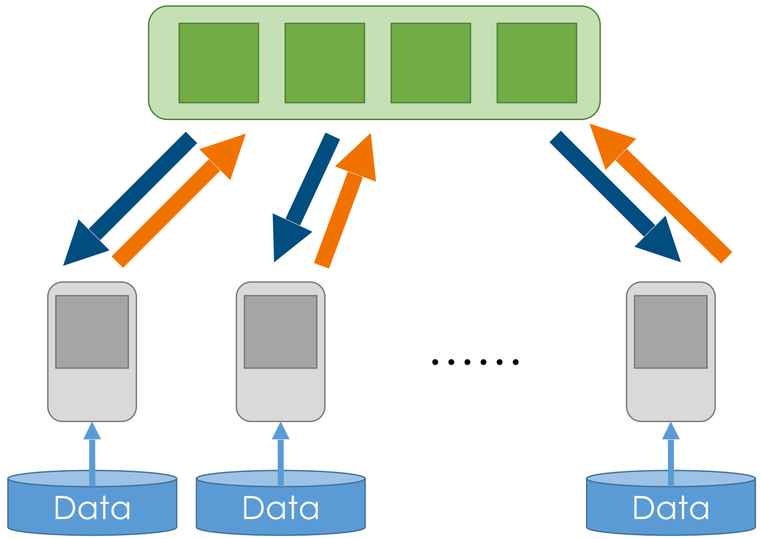
The communication schemes used in Parameter Server (PS)



Note: N is the number of training workers.

Communication Primitives

The communication schemes used in Parameter Server (PS)

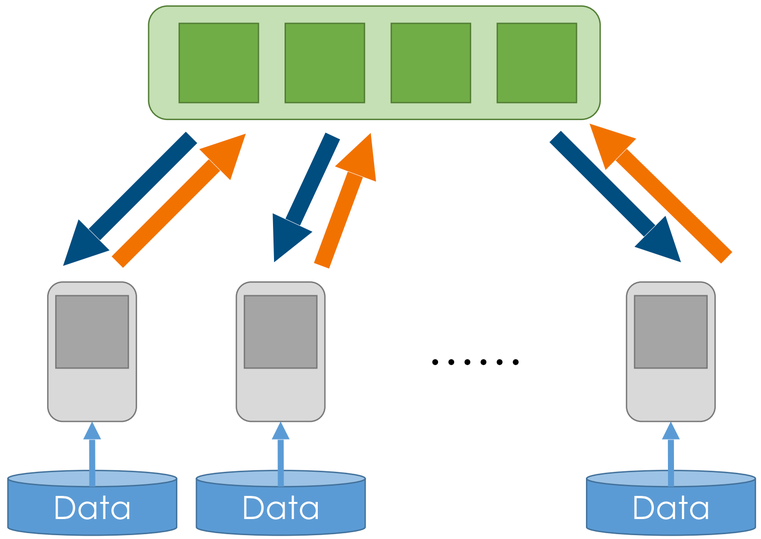


- Which type of communication scheme is used?
  - **Replicate & Pull**: ?
  - **Push & Sum**: ?

Note: N is the number of training workers.

Communication Primitives

The communication schemes used in Parameter Server (PS)

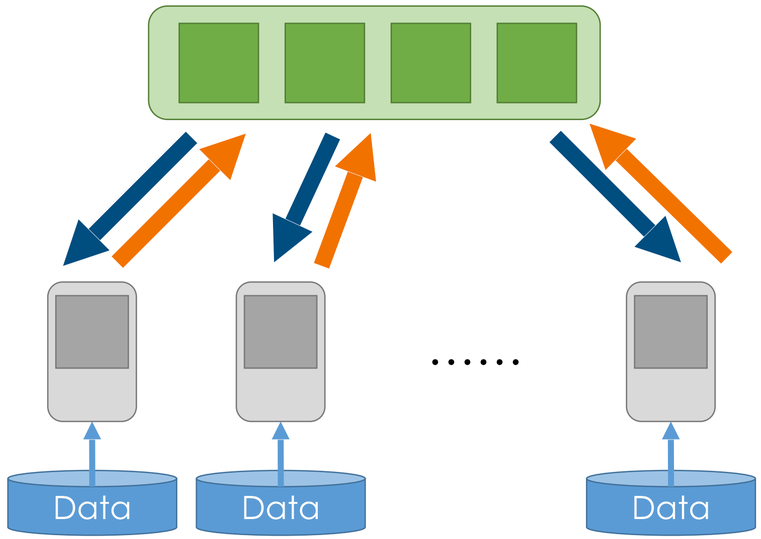


- Which type of communication scheme is used?
  - **Replicate & Pull**: Broadcast
    - Parameter Server to all workers
  - **Push & Sum**: Reduce
    - Average from all workers to Parameter Server

Note: N is the number of training workers.

Communication Primitives

The communication schemes used in Parameter Server (PS)

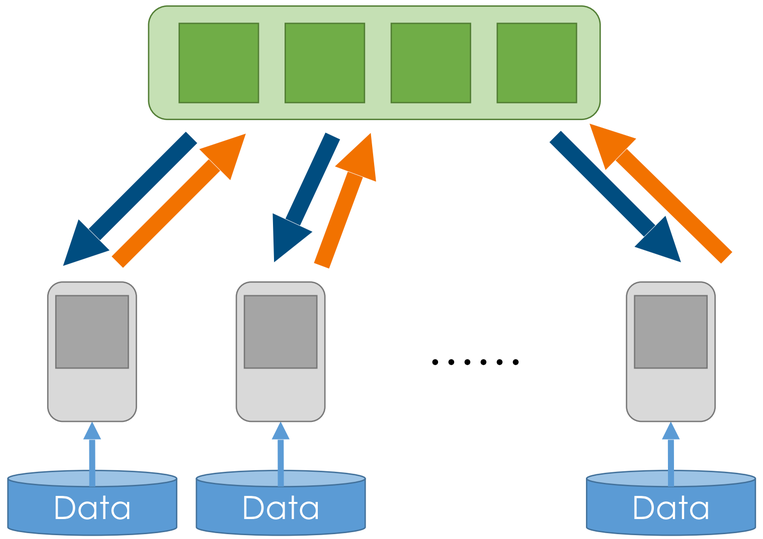


- Which type of communication scheme is used?
  - **Replicate & Pull**: Broadcast
    - Parameter Server to all workers
  - **Push & Sum**: Reduce
    - Average from all workers to Parameter Server
- What is the bandwidth requirements on each node?
  - **Replicate & Pull**:
  - **Push & Sum**:

Note: N is the number of training workers.

Communication Primitives

The communication schemes used in Parameter Server (PS)

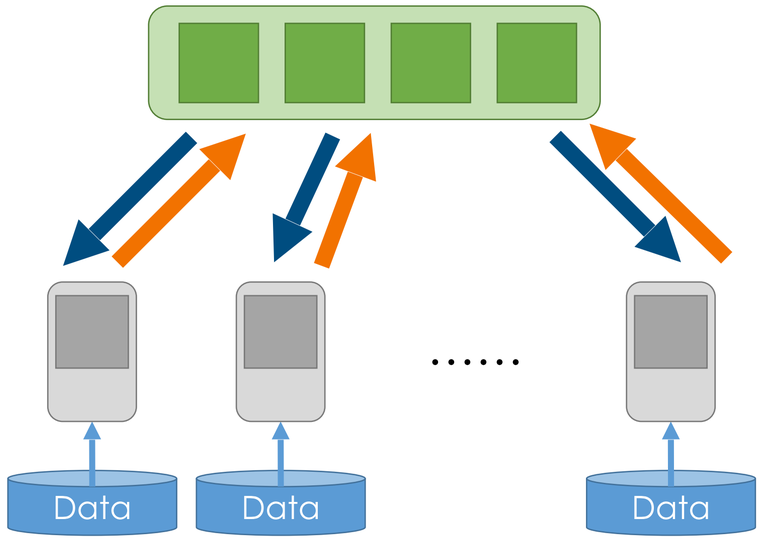


- Which type of communication scheme is used?
  - **Replicate & Pull**: Broadcast
    - Parameter Server to all workers
  - **Push & Sum**: Reduce
    - Average from all workers to Parameter Server
- What is the bandwidth requirements on each node?
  - **Replicate & Pull**:
    - Worker: O(1)
    - Parameter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

Note: N is the number of training workers.

Communication Primitives

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.

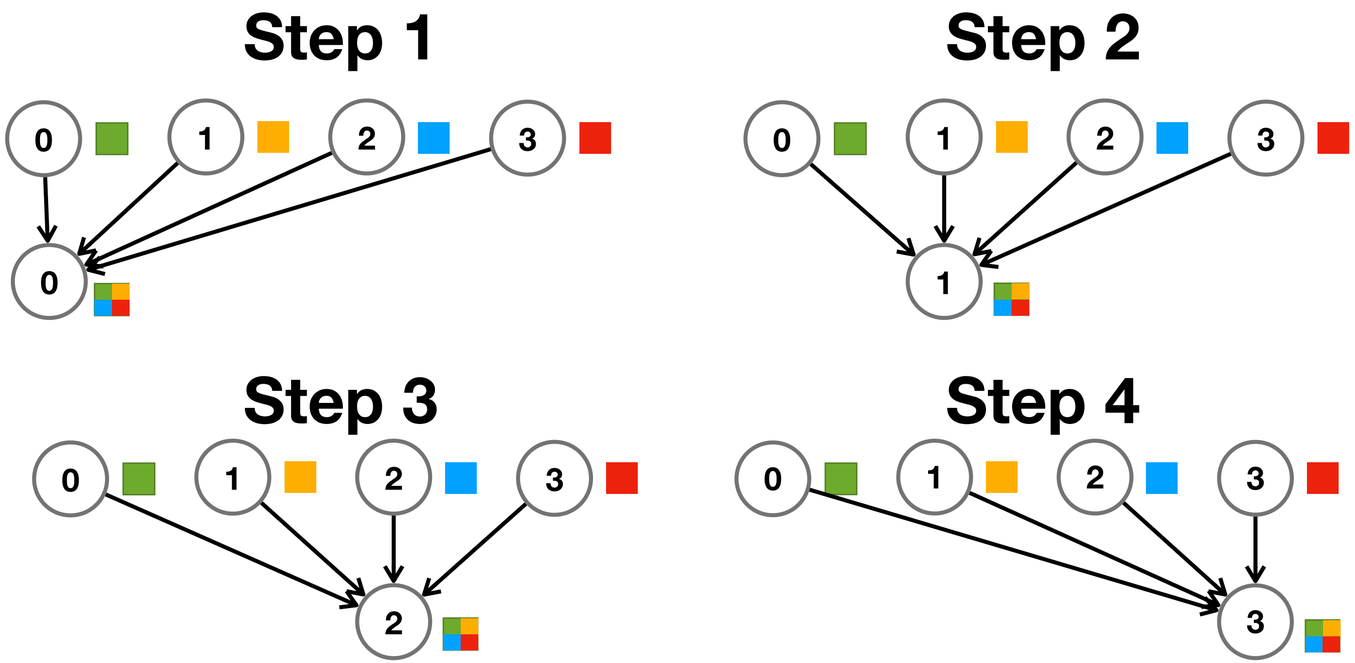
Can we perform the aggregation without a central server?

All-Reduce!

Note: N is the number of training workers.

Communication Primitives

Naive All-Reduce Implementation - Sequential




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

Time: O(N)

Bandwidth: O(N)

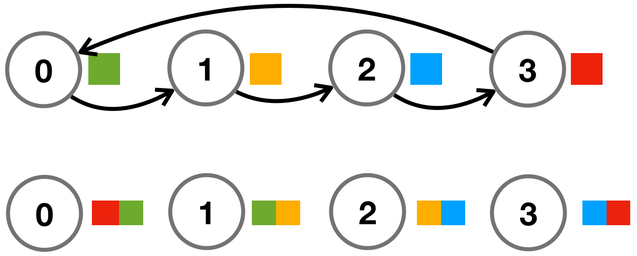
Each step performs a **SINGLE** Reduce operation

Note: N is the number of training workers.

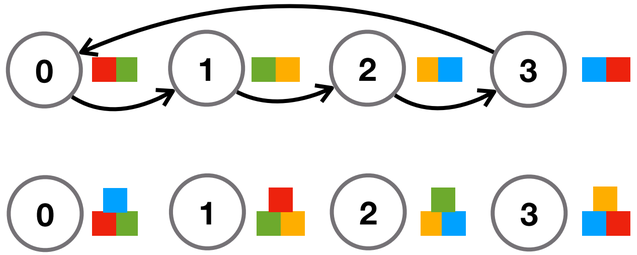
Communication Primitives

Better All-Reduce Implementation - Ring

Step 1



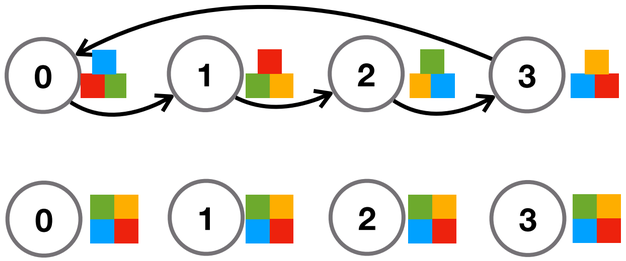
Step 2



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



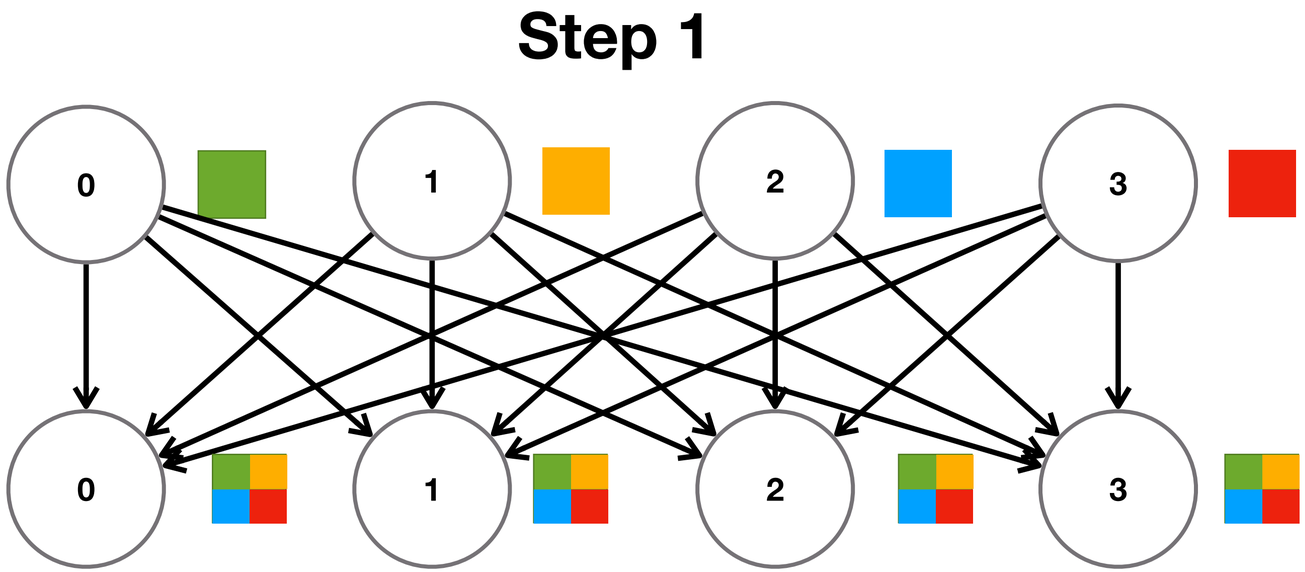
Time: O(N)

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

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

Communication Primitives

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.

Communication Primitives

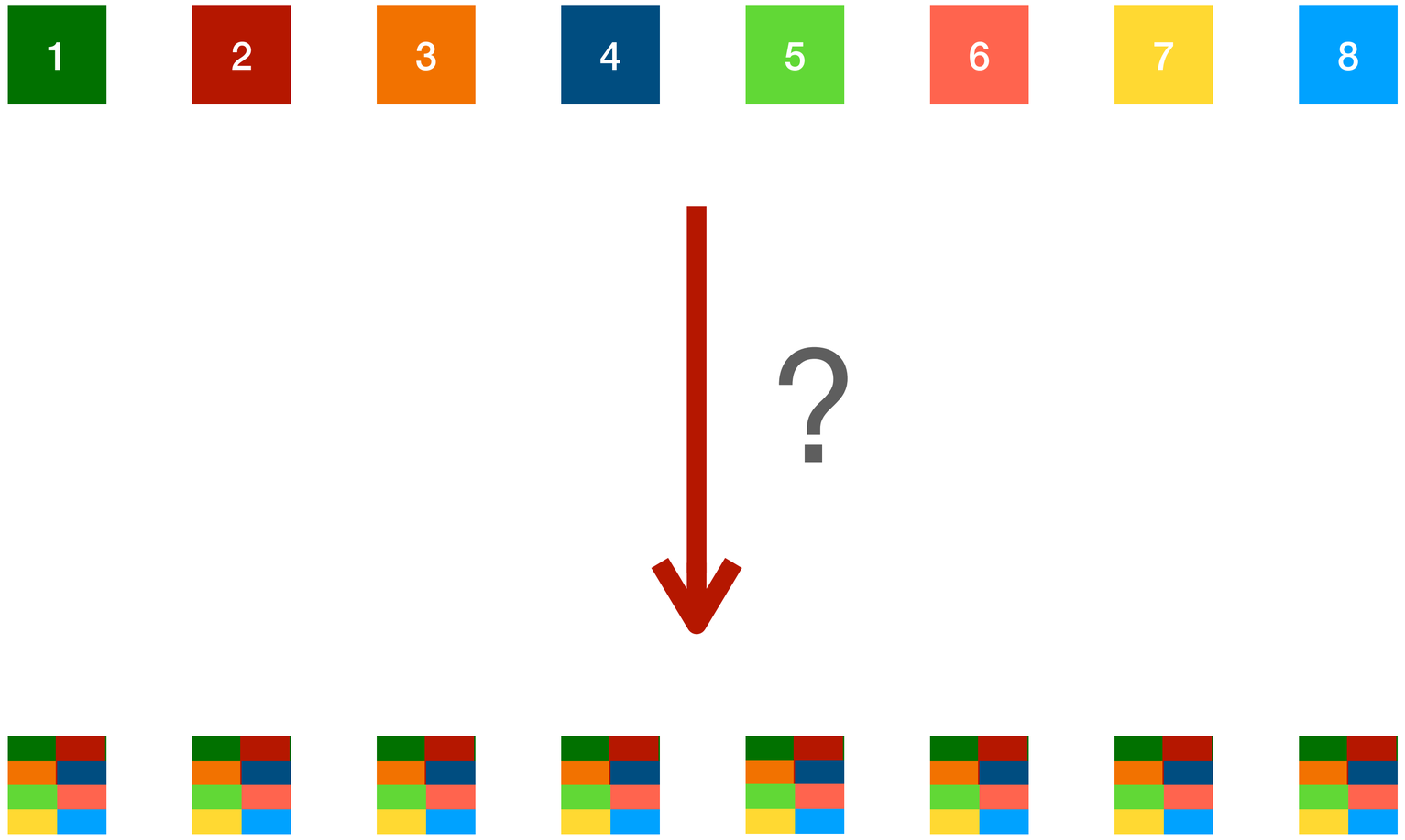
Compare Different Distributed Aggregation

	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?

Communication Primitives

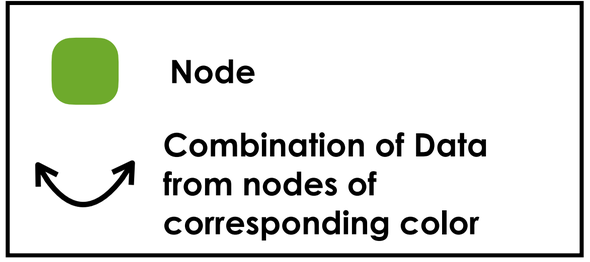
Recursive Halving All Reduce



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

Communication Primitives

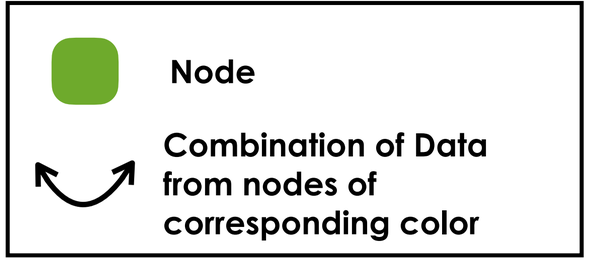
Recursive Halving All Reduce



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

Communication Primitives

Recursive Halving All Reduce

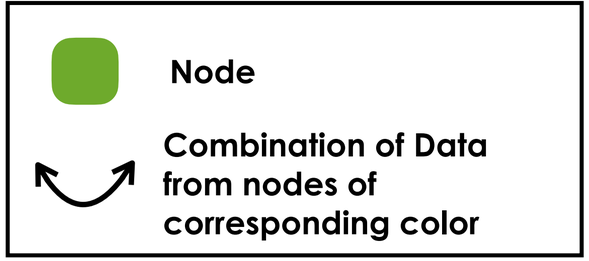


Step 1 - Each node exchanges with neighbors with offset 1

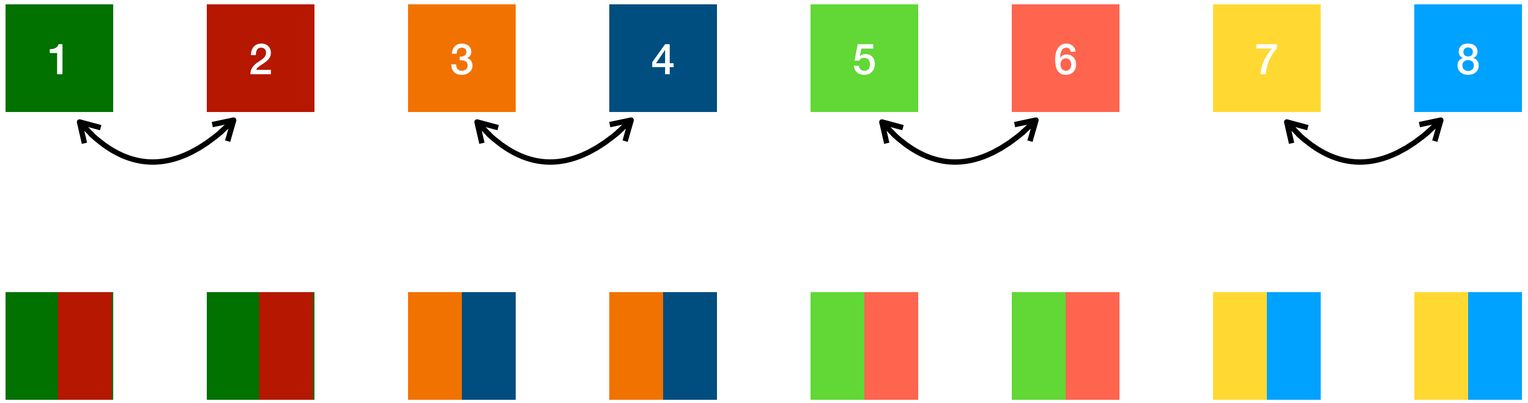
[1] Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. "Optimization of collective communication operations in MPICH."

Communication Primitives

Recursive Halving All Reduce



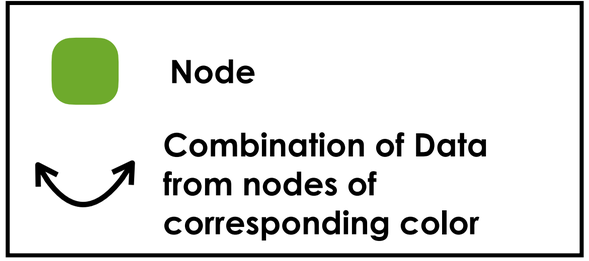
Step 1 - Each node exchanges with neighbors with offset 1



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

Communication Primitives

Recursive Halving All Reduce



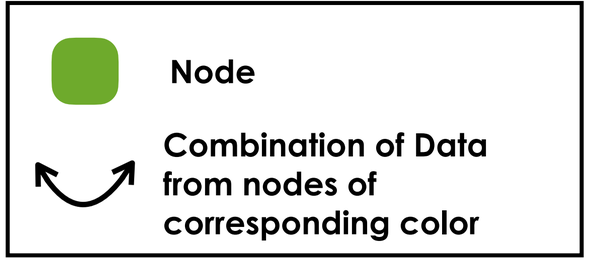
Step 1 - Each node exchanges with neighbors with offset 1

Step 2 - Each node exchanges with neighbors with offset 2

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

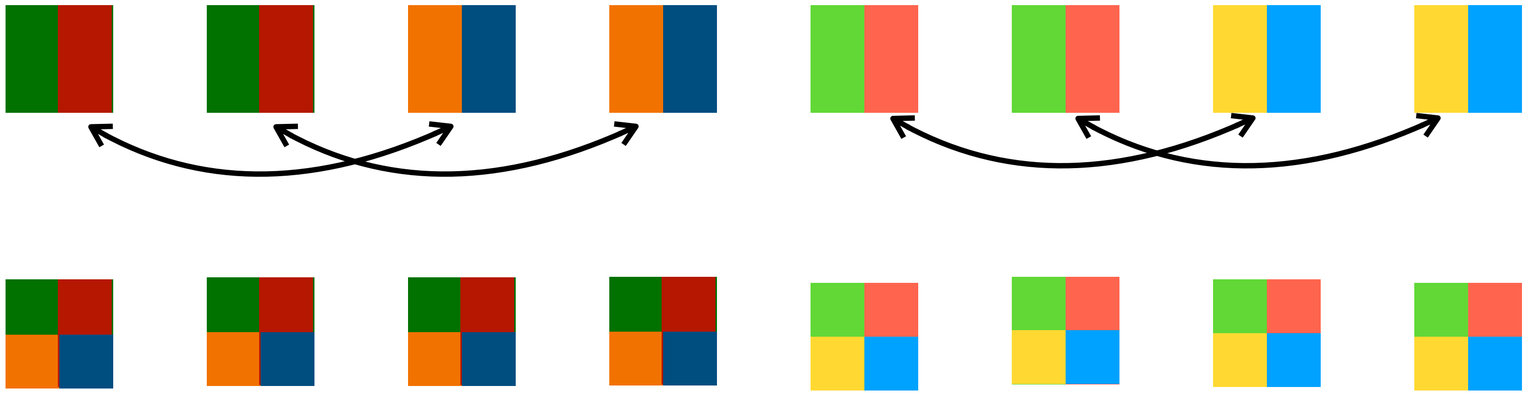
Communication Primitives

Recursive Halving All Reduce



Step 1 - Each node exchanges with neighbors with offset 1

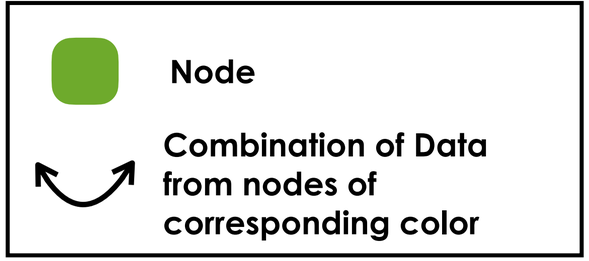
Step 2 - Each node exchanges with neighbors with offset 2



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

Communication Primitives

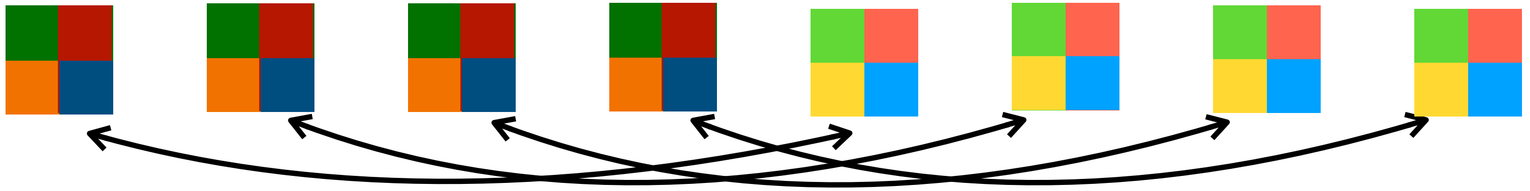
Recursive Halving All Reduce



Step 1 - Each node exchanges with neighbors with offset 1

Step 2 - Each node exchanges with neighbors with offset 2

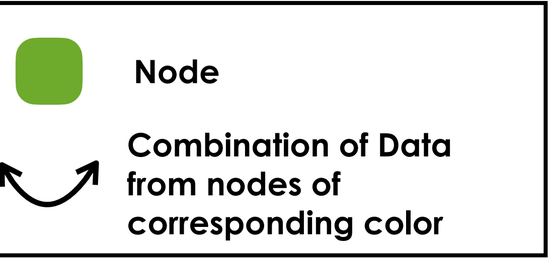
Step 3 - Each node exchanges with neighbors with offset 4



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

Communication Primitives

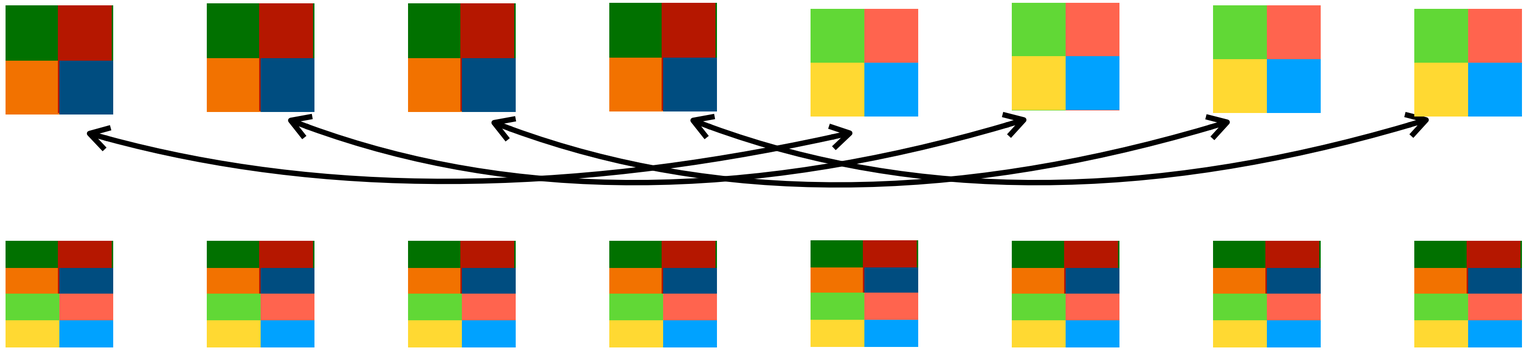
Recursive Halving All Reduce



Step 1 - Each node exchanges with neighbors with offset 1

Step 2 - Each node exchanges with neighbors with offset 2

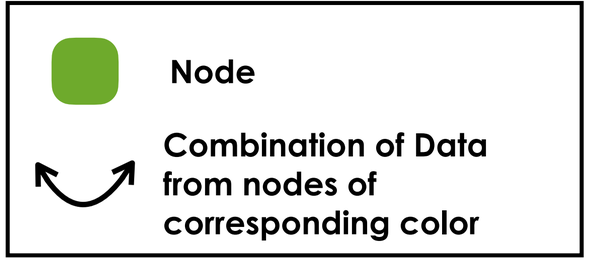
Step 3 - Each node exchanges with neighbors with offset 4



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

Communication Primitives

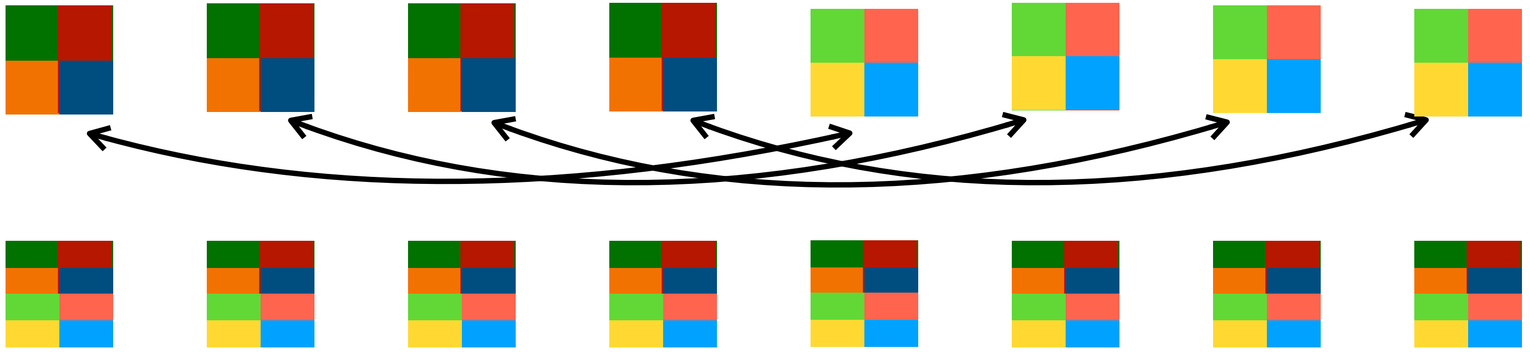
Recursive Halving All Reduce



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.

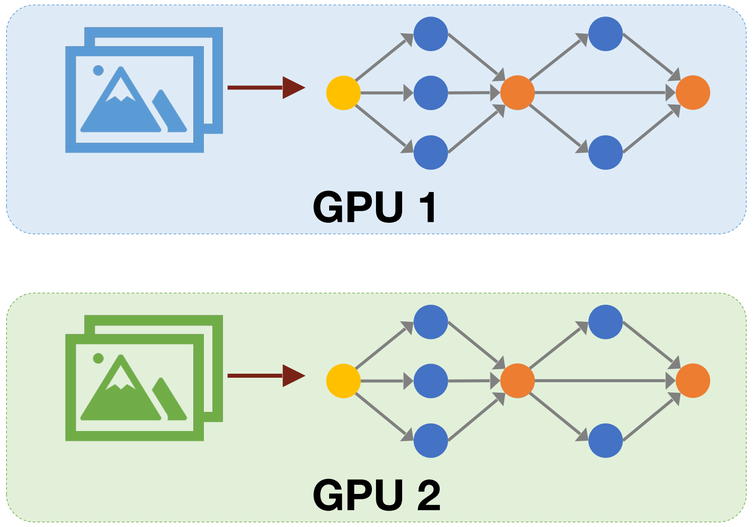
[1] Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. "Optimization of collective communication operations in MPICH."

Lecture Plan

Understand how distributed training works and improve the efficiency

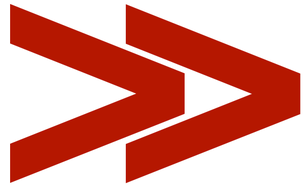
1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. **Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP**
6. Pipeline parallelism
7. Tensor parallelism

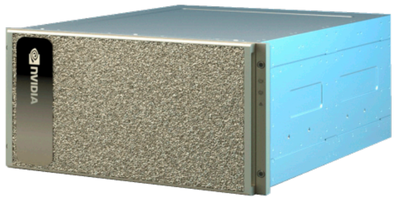
ZeRO-1 / 2 / 3 and FSDP



If we train a super-large model (e.g., GPT-3 175B):

175B * 2 Bytes (fp16)
= 350GB Memory





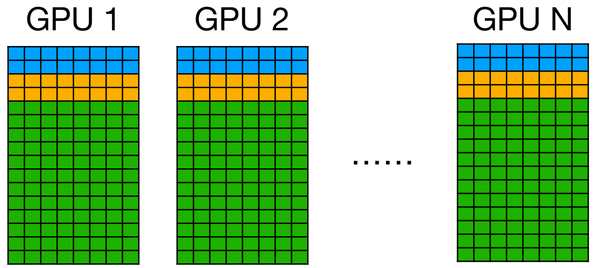
Nvidia A100 80GB

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

Further, training requires to store gradients and optimizer states.

ZeRO-1 / 2 / 3 and FSDP

DeepSpeed Zero to Optimize Memory



Baseline

: weights, 2 bytes

: gradients, 2 bytes

: optimizer states, K bytes for Adam

* K=12 from ZeRO: “fp32 copy of the parameters, momentum and variance”

N: number of GPUs ψ: number of parameters

- Naive data parallelism has N copies of weights, gradients and optimizer states.
- This introduces redundancy and easily leads to OOM when training large models.

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2\text{bytes}}_{\text{gradients}} + \underbrace{12\text{bytes}}_{\text{optim states}})\psi$$

- Even for most most advanced A100/H100 with 80GB memory, the largest trainable model is

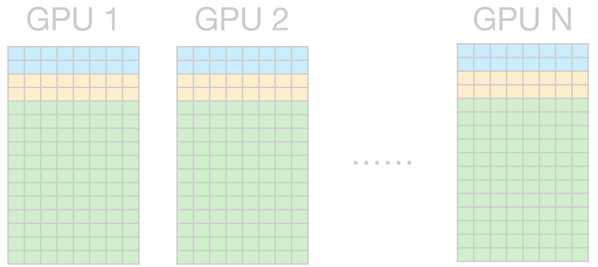
$$80\text{GB}/16\text{Bytes} = 5.0B$$

which is far away from current SOTA (175B).

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

ZeRO-1 / 2 / 3 and FSDP

DeepSpeed Zero to Optimize Memory



Baseline

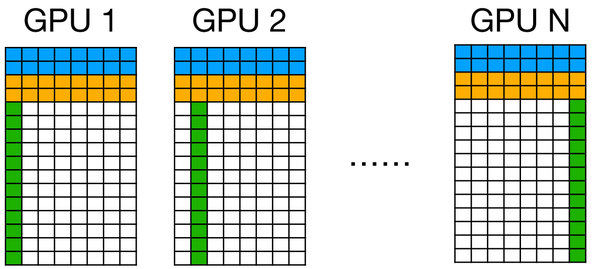
: weights, 2 bytes

: gradients, 2 bytes

: optimizer states, K bytes for Adam

* K=12 from ZeRO: "fp32 copy of the parameters, momentum and variance"

N: number of GPUs ψ: number of parameters



ZeRO-1

- Zero-1 proposes to partition / shard **optimizer states**.

Memory Consumption

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

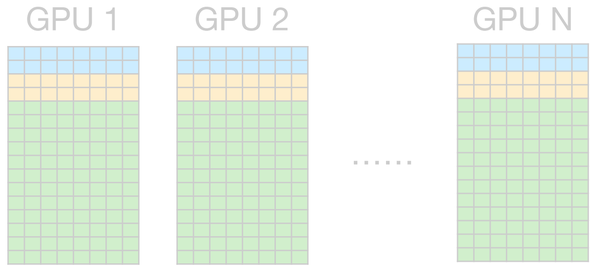
Largest Model Size for Training (N=64)

$$80\text{GB}/4.2\text{Bytes} = 19B$$

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

ZeRO-1 / 2 / 3 and FSDP

DeepSpeed Zero to Optimize Memory



Baseline

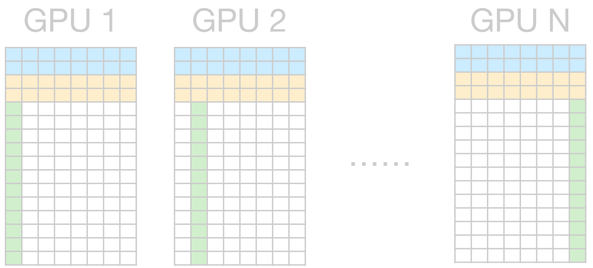
: weights, 2 bytes

: gradients, 2 bytes

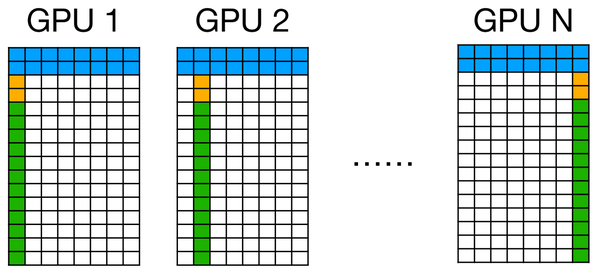
: optimizer states, K bytes for Adam

* K=12 from ZeRO: "fp32 copy of the parameters, momentum and variance"

N: number of GPUs ψ: number of parameters



ZeRO-1



ZeRO-2

- Zero-2 proposes to partition / shard **optimizer states** and **gradients**.

Memory Consumption

$$\left(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}}\right)\psi$$

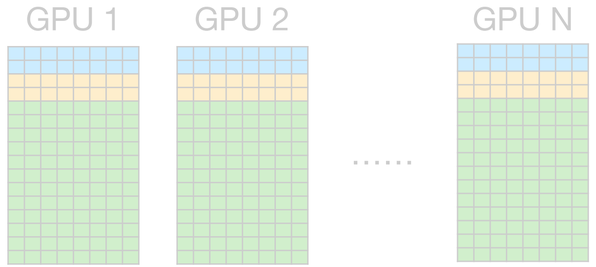
Largest Model Size for Training (N=64)

$$80\text{GB}/2.2\text{Bytes} = 36B$$

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

ZeRO-1 / 2 / 3 and FSDP

DeepSpeed Zero to Optimize Memory



Baseline

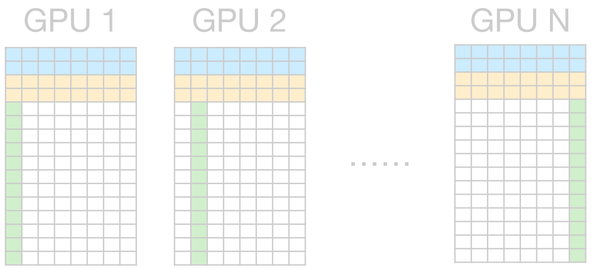
: weights, 2 bytes

: gradients, 2 bytes

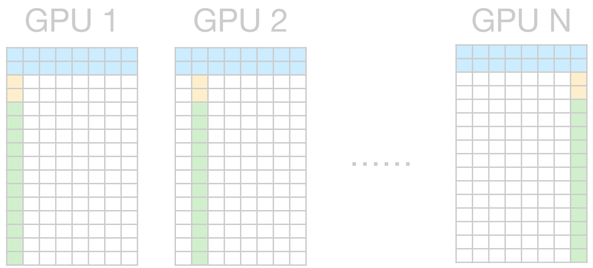
: optimizer states, K bytes for Adam

* K=12 from ZeRO: "fp32 copy of the parameters, momentum and variance"

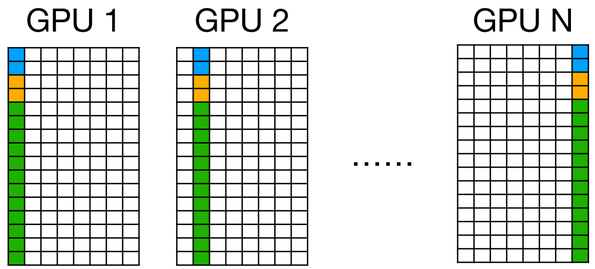
N: number of GPUs ψ: number of parameters



ZeRO-1



ZeRO-2



ZeRO-3

- Zero-3 proposes to partition / shard **optimizer states**, **gradients** and **weights**.

Memory Consumption

$$(\underbrace{2/N\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

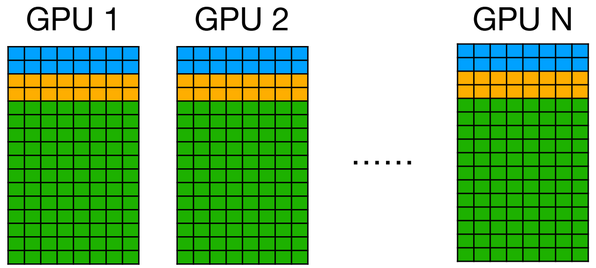
Largest Model Size for Training (N=64)

$$80\text{GB}/0.25\text{Bytes} = 320B$$

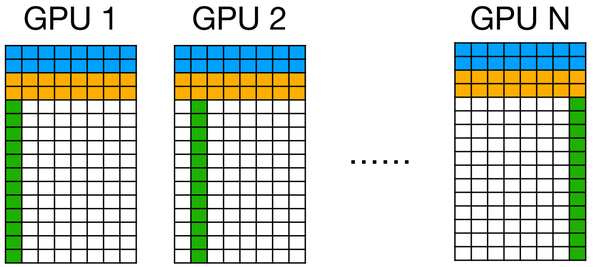
ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

ZeRO-1 / 2 / 3 and FSDP

Compare Zero-1/2/3; FSDP



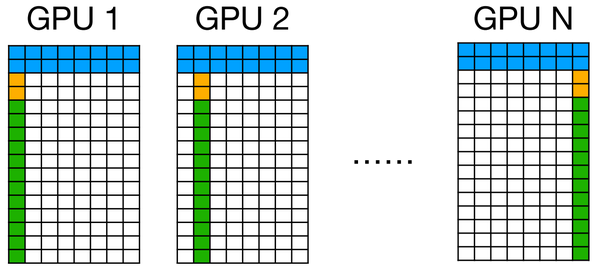
$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2\text{bytes}}_{\text{gradients}} + \underbrace{12\text{bytes}}_{\text{optim states}})\psi$$



ZeRO-1

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

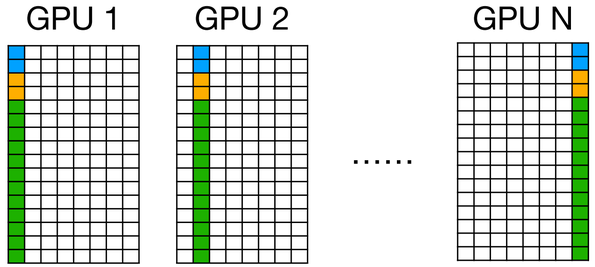
$$80\text{GB}/4.2\text{Bytes} = 19B$$



ZeRO-2

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

$$80\text{GB}/2.2\text{Bytes} = 36B$$



ZeRO-3

$$(\underbrace{2/N\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

$$80\text{GB}/0.25\text{Bytes} = 320B$$

In PyTorch, ZeRO-3 is implemented via `FullyShardedDataParallel`, known as FSDP.

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

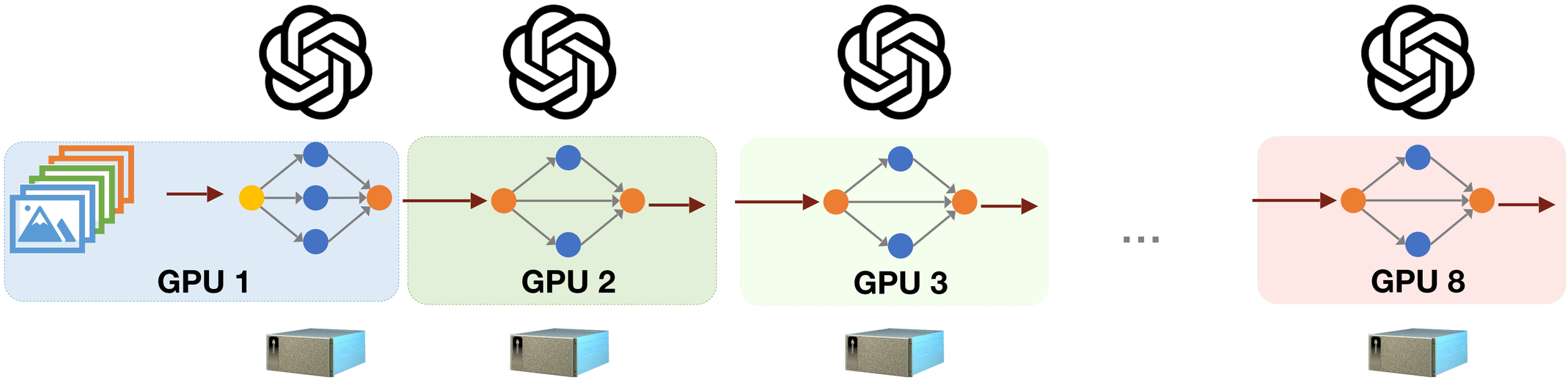
Lecture Plan

Understand how distributed training works and improve the efficiency

1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. **Pipeline parallelism**
7. Tensor parallelism

Pipeline Parallelism

Instead of splitting the data, pipeline parallelism splits the model

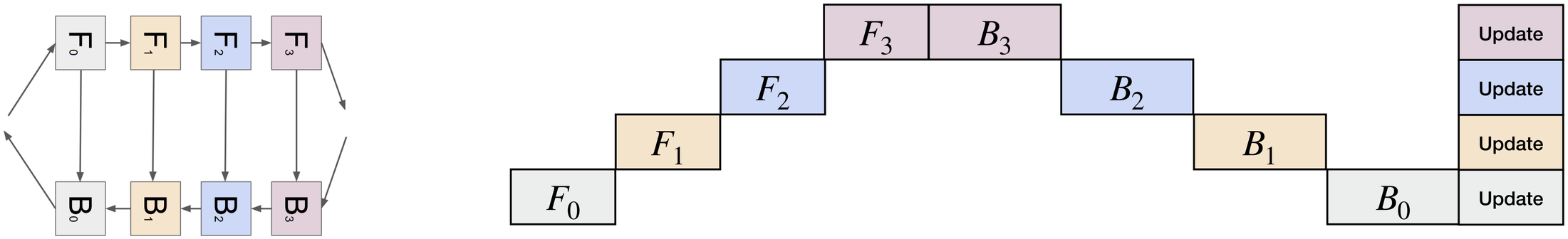


350GB / 8 cards = 43.75G < 80G

With Pipeline Parallelism, large ML models can be placed and trained on GPUs.

Pipeline Parallelism Workflow

Naive Implementation



(a). Training data flow (b). Training timeline

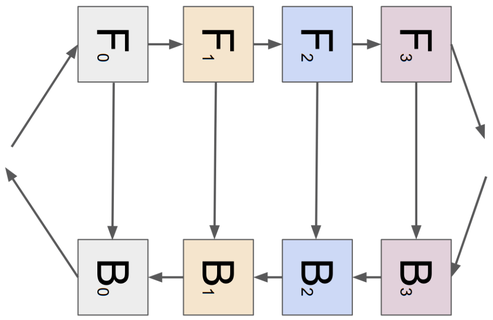
F: Forward B: Backward. Train a 4 layer network with Pipeline Parallelism.

Pipeline Parallelism is needed for training a bigger DNN model by dividing the model into partitions and assigning different partitions to different accelerators.

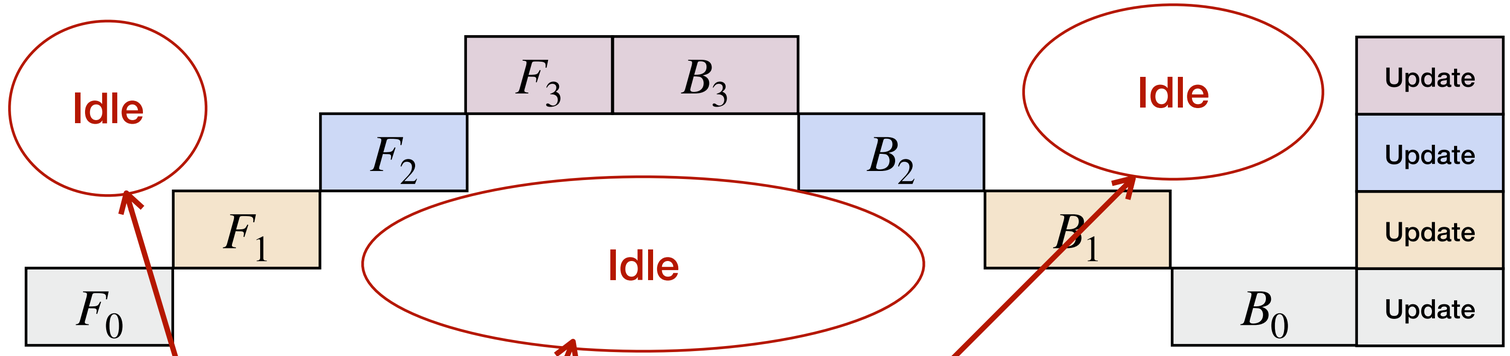
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Pipeline Parallelism Workflow

Naive Implementation



(a). Training data flow



(b). Training timeline

F: Forward B: Backward. Train a 4 layer network with Pipeline Parallelism.

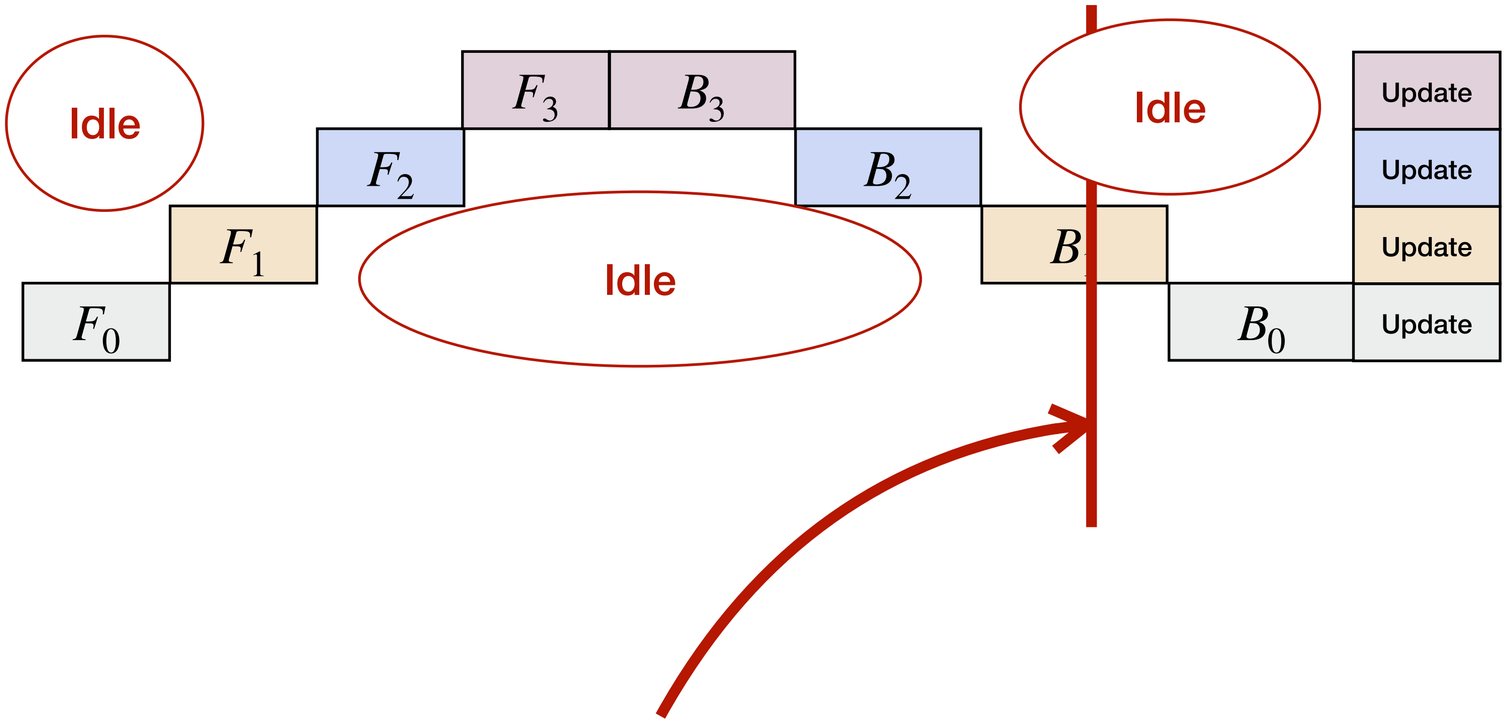
Pipeline Parallelism is needed for training a bigger DNN model by dividing the model into partitions and assigning different partitions to different accelerators.

But accelerators are significantly **under-utilized** during Pipeline Parallelism!

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

Pipeline Parallelism Workflow

Naive Pipeline Parallelism Suffers from Utilization



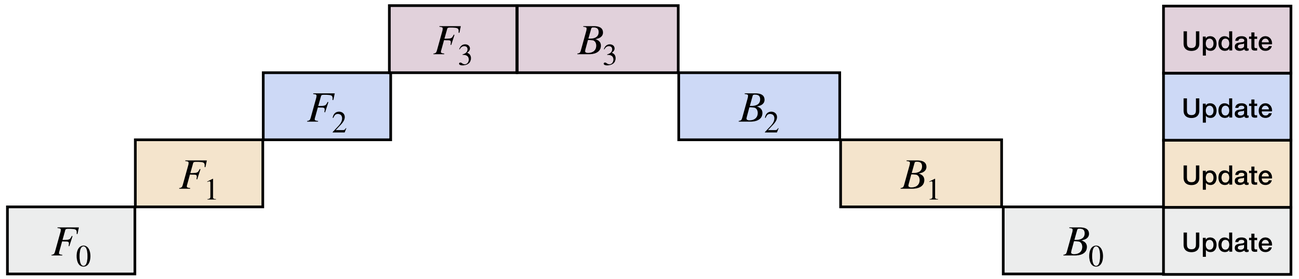
Only one device is computing at a time and others are waiting for it.

Theoretical utilization (in this 4 layer NN): 25% (**low!**)

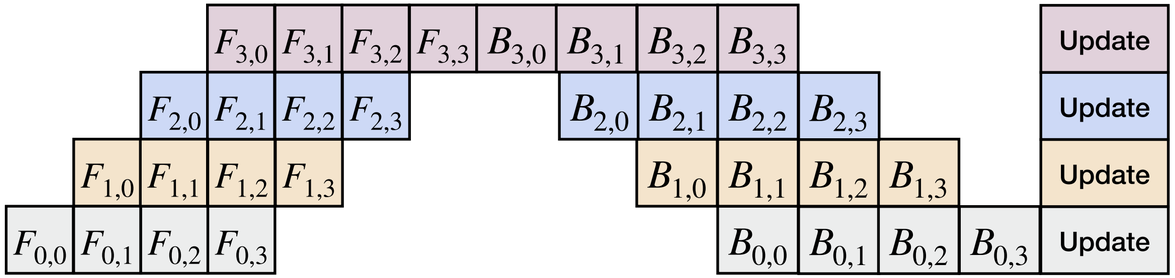
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Pipeline Parallelism Workflow

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



(a). Naive Pipeline Parallelism



(b). Improved Pipeline parallelism

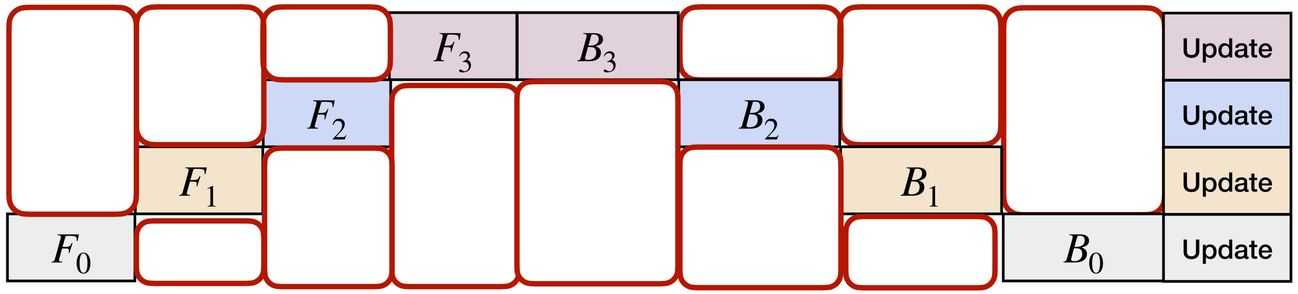
- Split a single batch to micro batches
  - [16, 10, 512] ->
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]

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

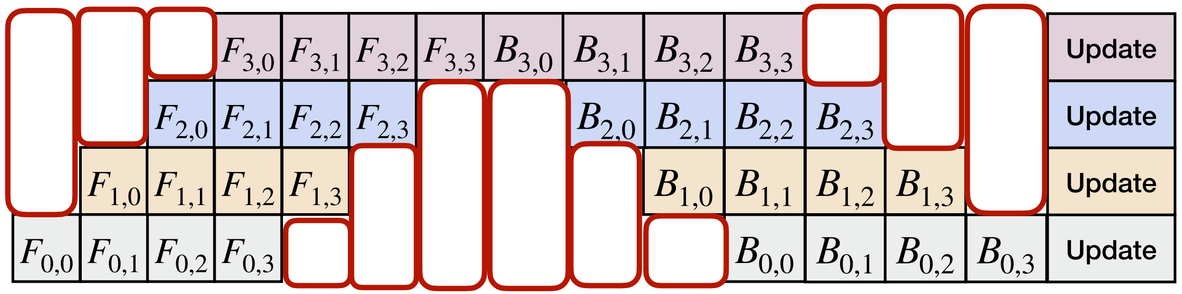
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Pipeline Parallelism Workflow

Micro-batch improvs the device utilization



(a). Naive Pipeline Parallelism



(b). Pipeline parallelism

Utilization: (25%)

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

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

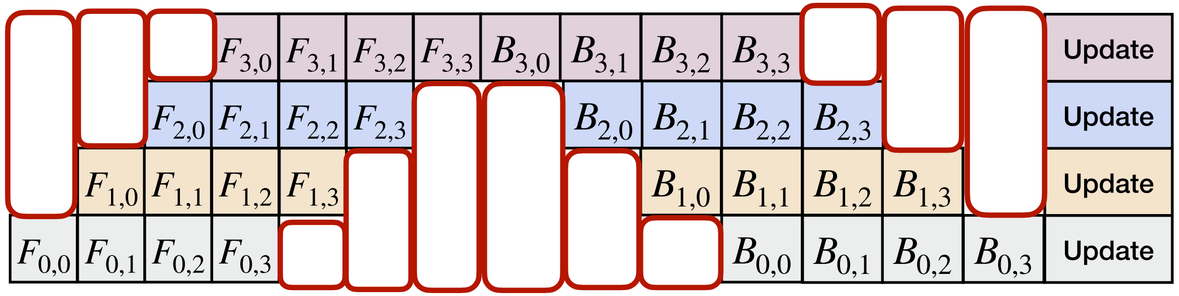
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]

Lecture Plan

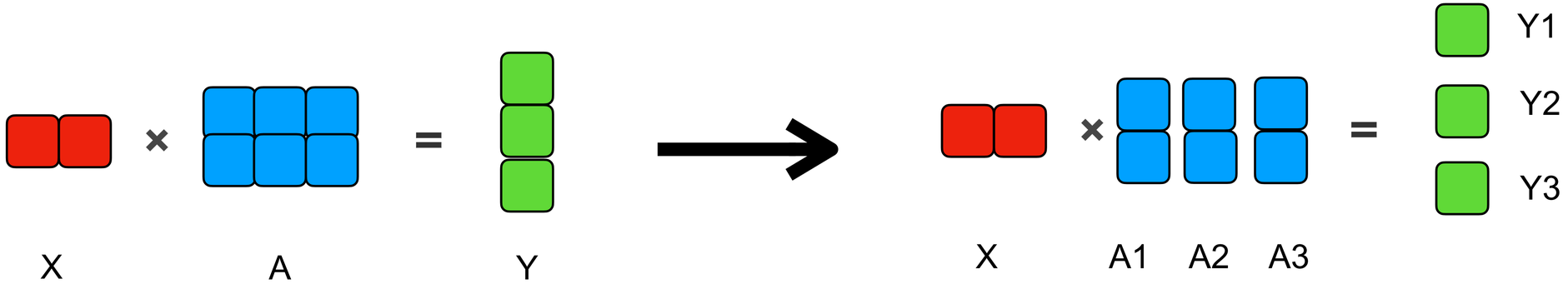
Understand how distributed training works and improve the efficiency

1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. **Tensor parallelism**

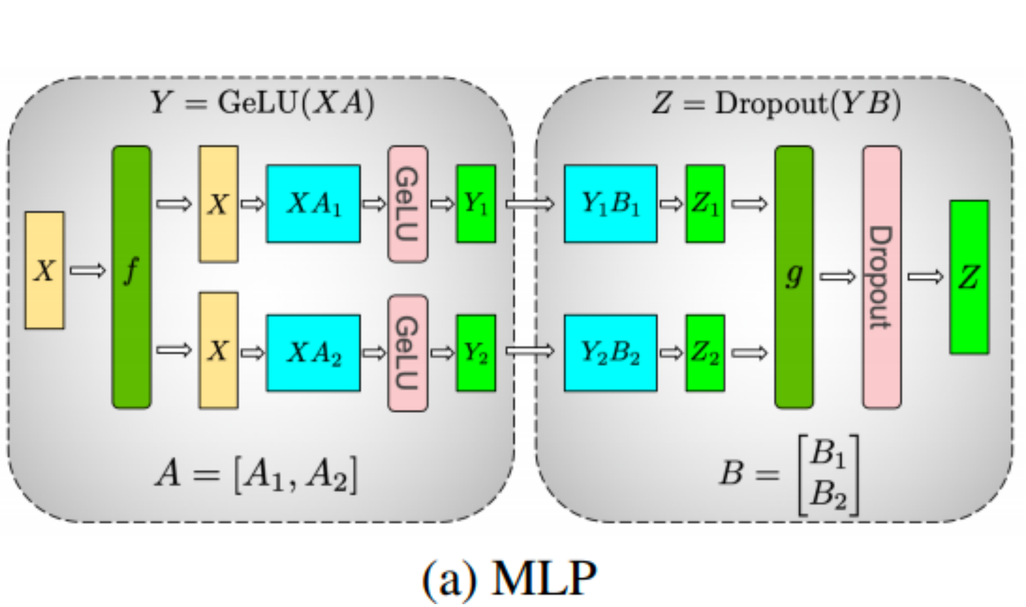
Tensor Parallelism



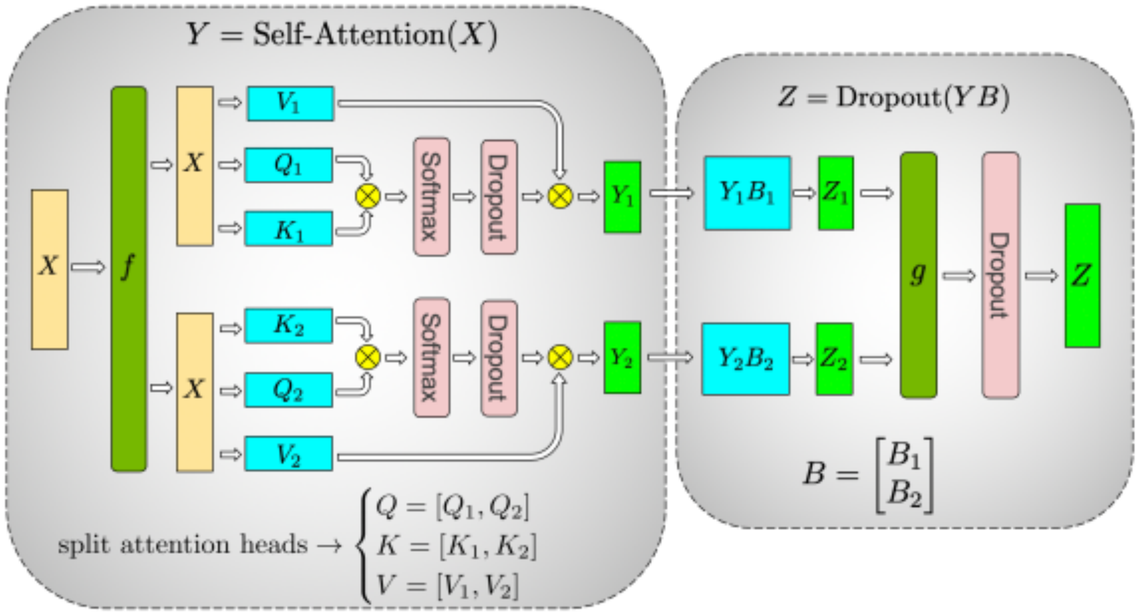
- The optimized pipeline parallelism still leads to idle time of GPUs, can we further improve it (while keeping memory usage low)?
- Make the model partition ***more fine-grained***!



Tensor Parallelism



(a) MLP



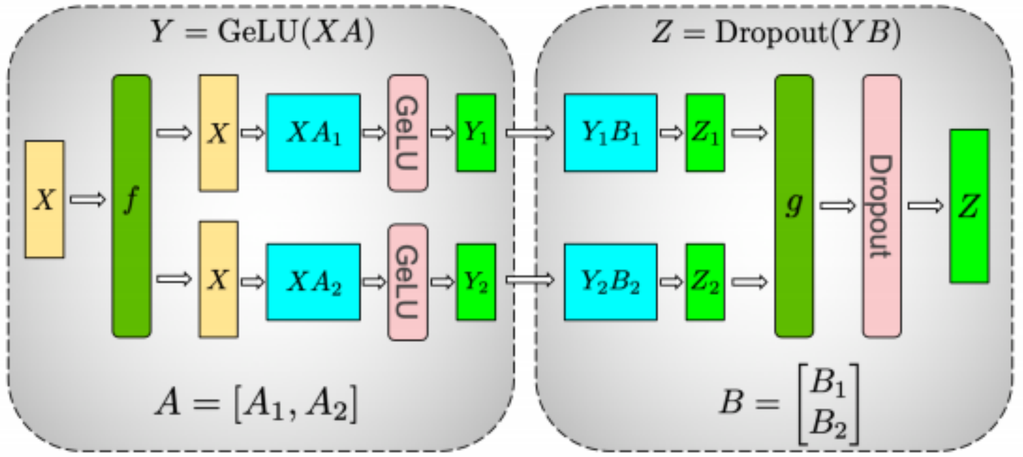
(b) Self-Attention

- Tensor parallelism: split a weight tensor into N chunks and parallelize
- f and g denote the AllReduce / Identity operations to synchronize.
- The params and activations (blue part) are spliced across different GPUs.

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

Tensor Parallelism

Partition in First FFN Layer

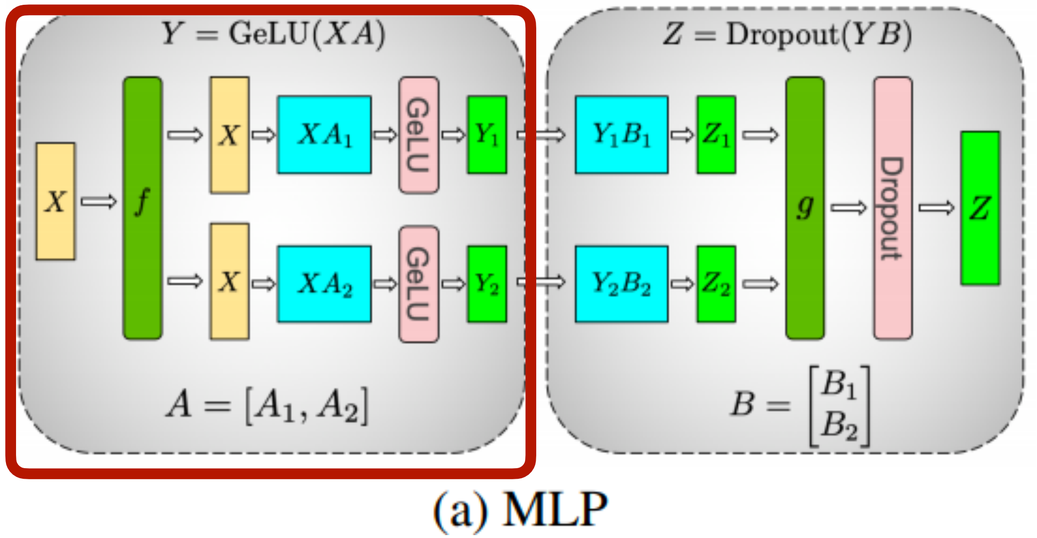


(a) MLP

- Take [1, 2] @ [2, 6] @ [6, 2] as an example

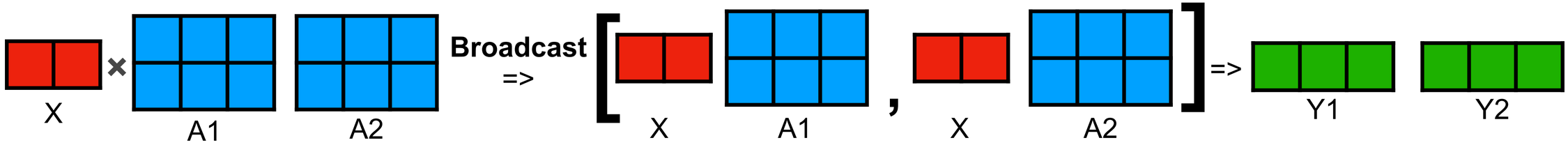
Tensor Parallelism

Partition in First FFN Layer



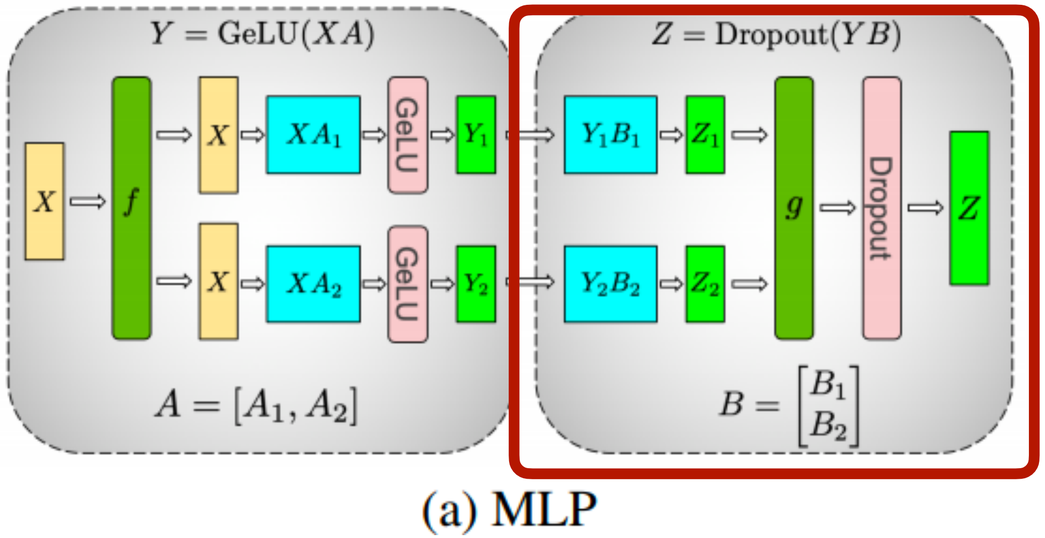
(a) MLP

- Take [1, 2] @ [2, 6] @ [6, 2] as an example
- The first Linear layer is partitioned in **column** parallel fashion.
- We use **Broadcast to duplicate input x** to all devices.



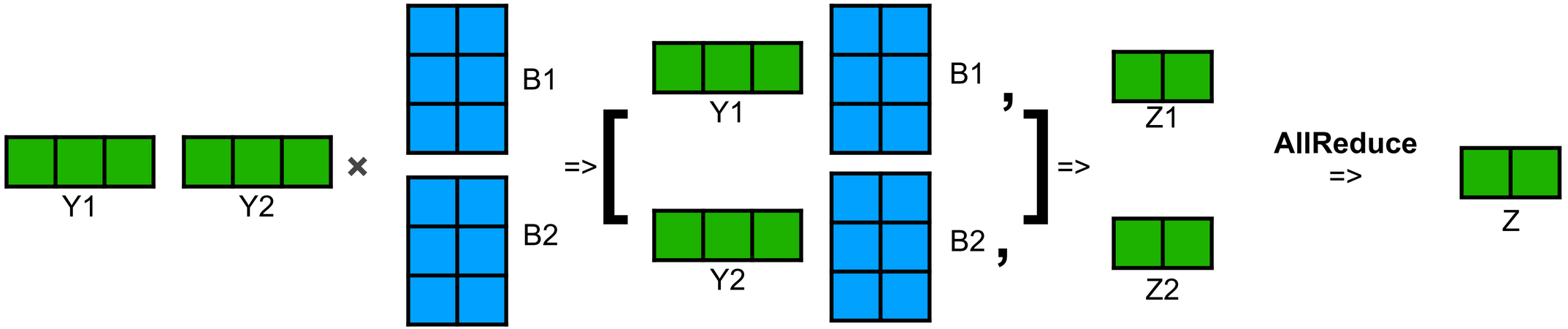
Tensor Parallelism

Partition in Second FFN Layer



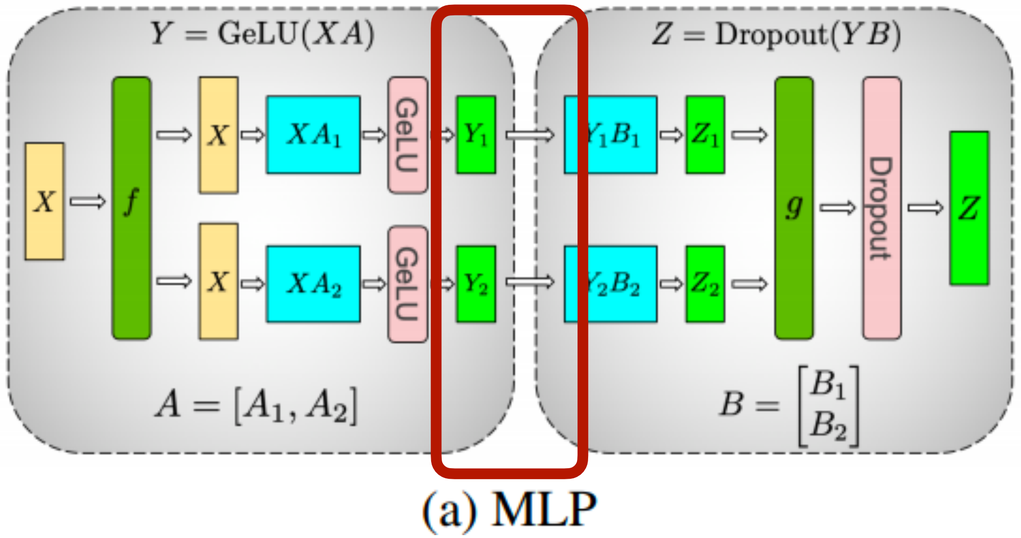
(a) MLP

- Take [1, 2] @ [2, 6] @ [6, 2] as an example
- The second Linear layer is partitioned in **row** parallel fashion.
- We **AllReduce** the output to gather outputs.



Tensor Parallelism

Partition in FFN Layers



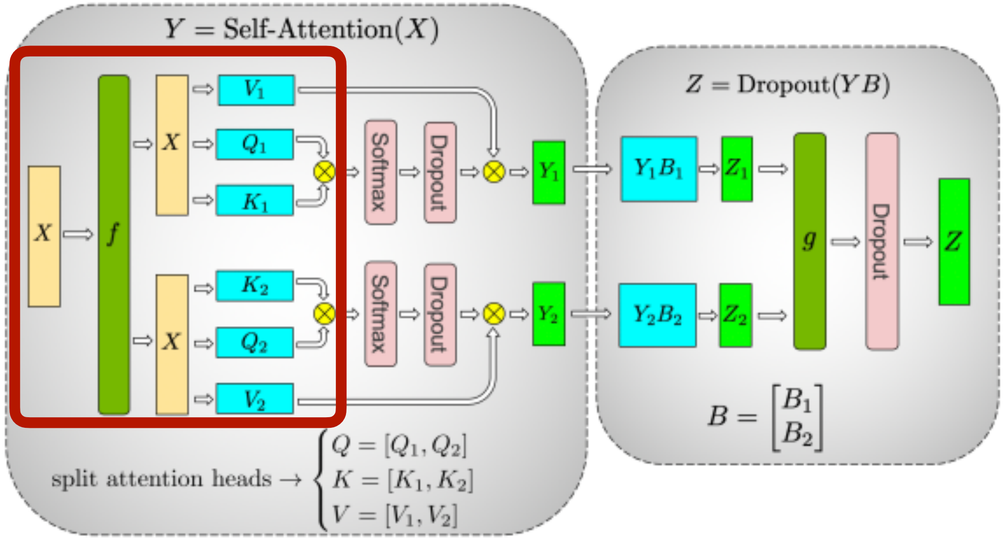
(a) MLP

- Partitioning the **first** GEMM in **column** and split the **second** GEMM along **rows**
- Thus it takes the output directly without requiring any communication
- As long as the communication is not bottleneck, GPUs can be fully utilized.

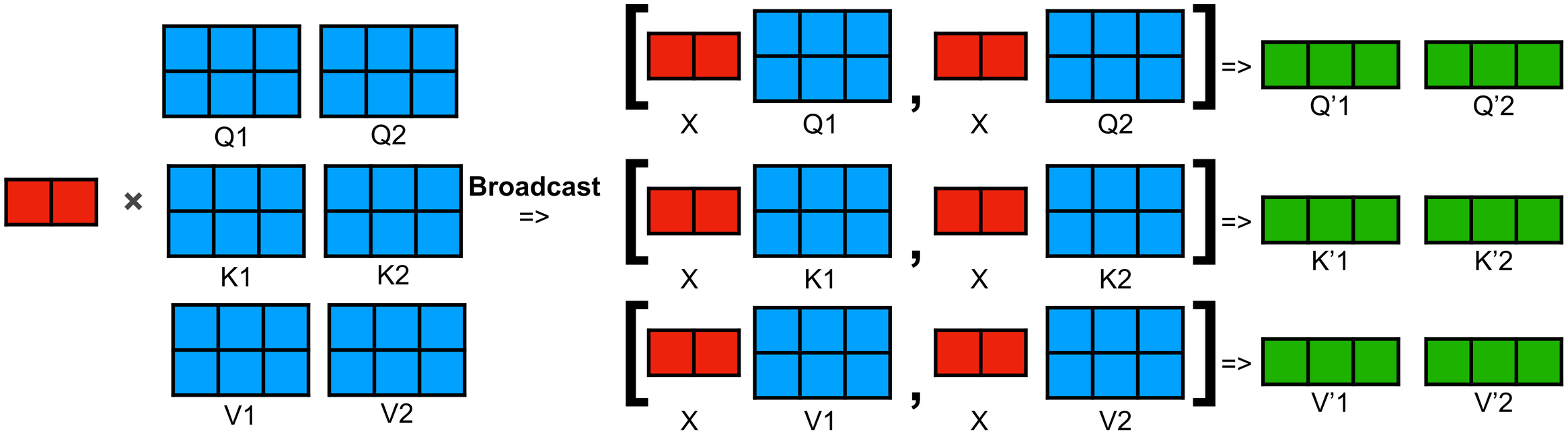
Tensor Parallelism

Partition Attention Layers: QKV Projection

- Split the QKV projection layer via **columns**.
- Input X is **Broadcasted** to all devices.



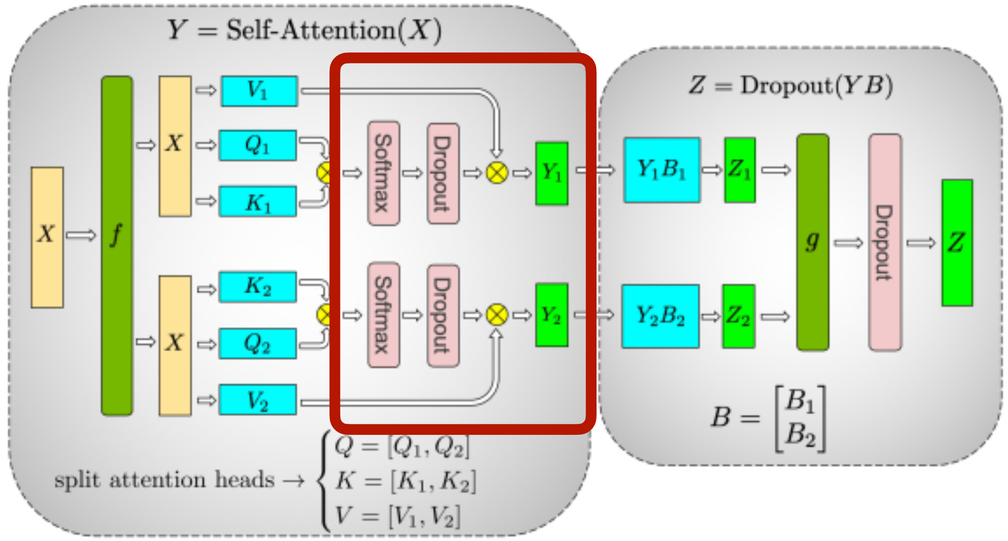
(b) Self-Attention



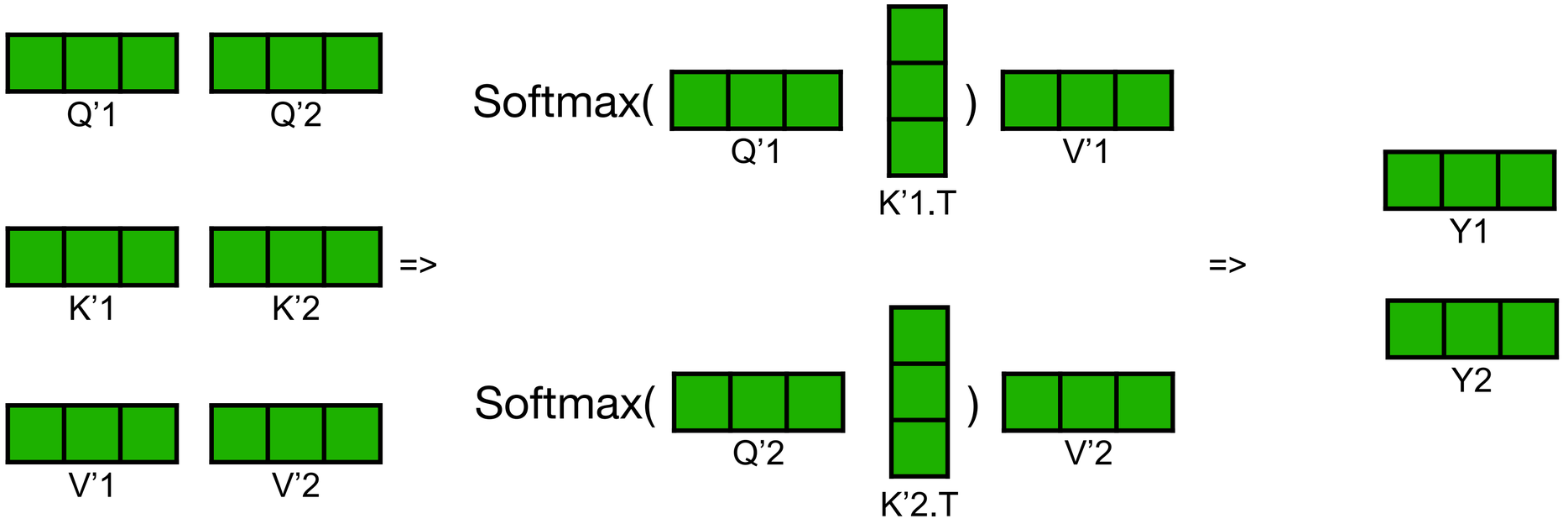
Tensor Parallelism

Partition Attention Layers: Attention

- Soft(Q@K.T)@V is performed locally with no extra communication



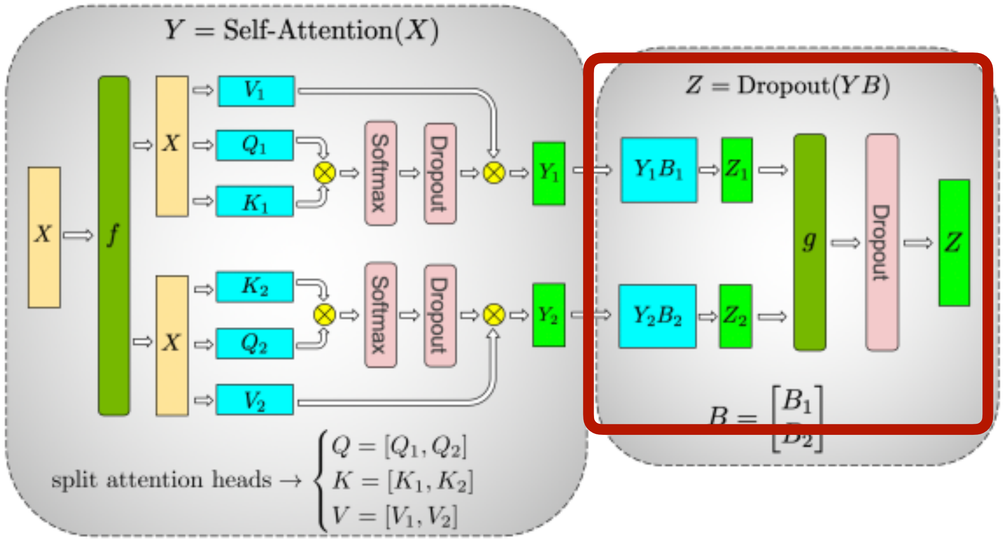
(b) Self-Attention



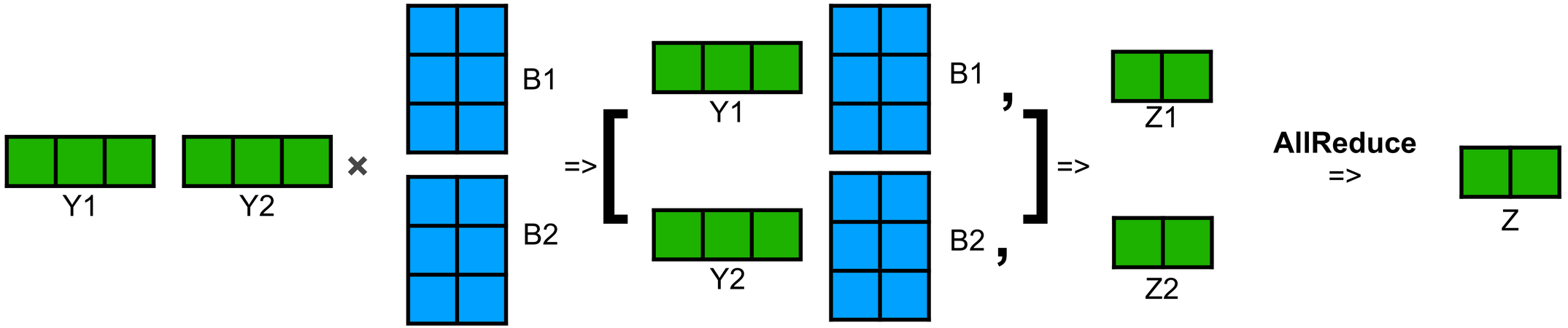
Tensor Parallelism

Partition Attention Layers: Out Projection

- OutputProjection layer is splitted by **rows**.
- The computation can be done without synchronization.
- The outputs is then collected with **AllReduce**.

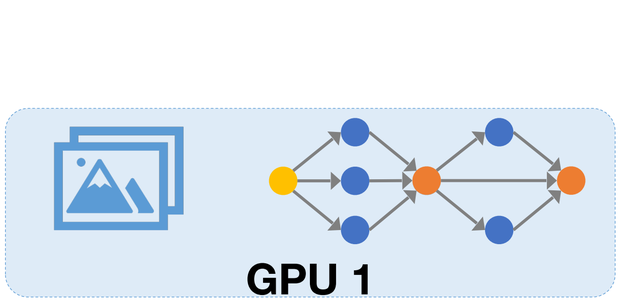


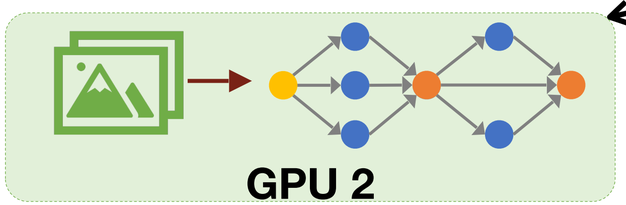
(b) Self-Attention

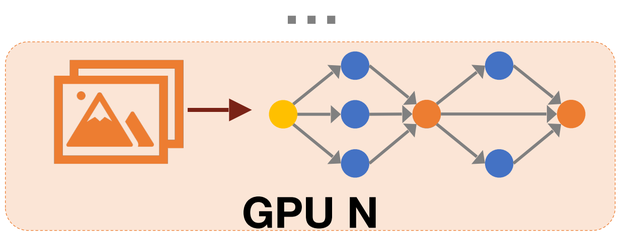


Summary of Today's Lecture

- Different parallelisms.

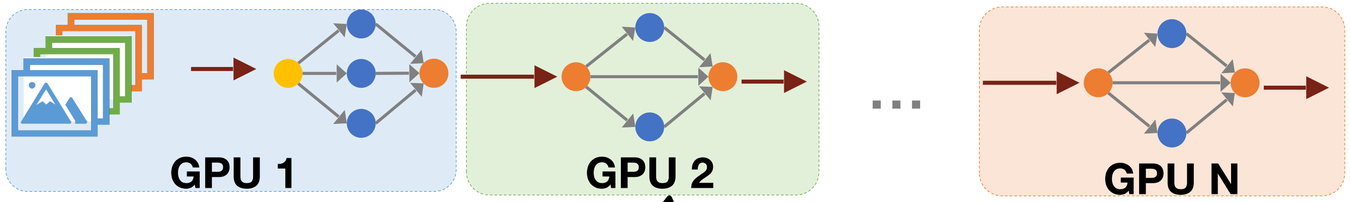






Data Parallelism:

- Split the data

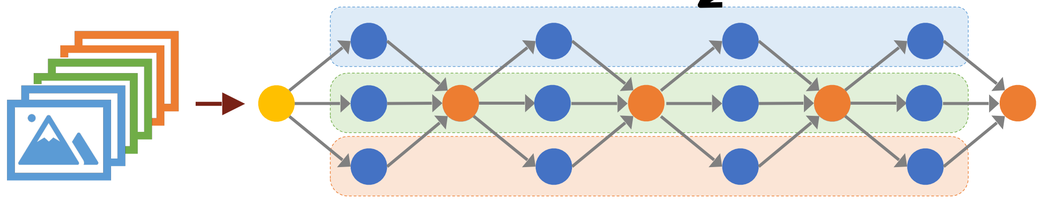


Pipeline Parallelism:

- Split the model by layer

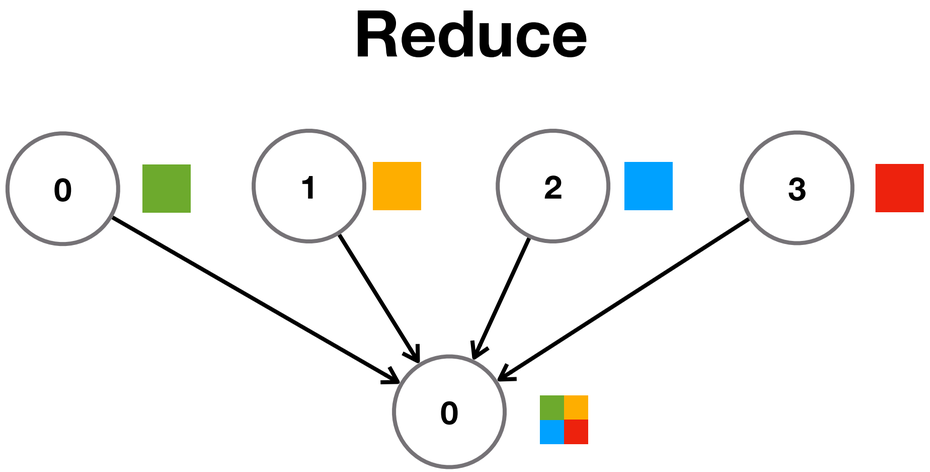
Tensor Parallelism:

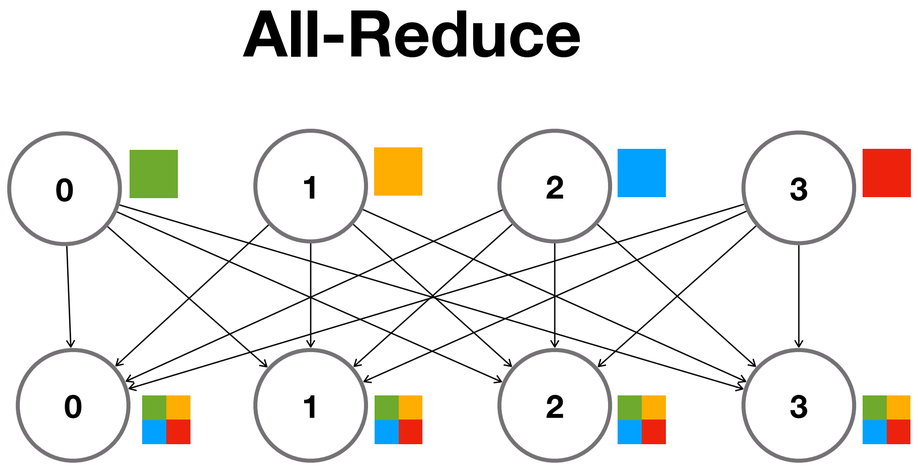
- Split the model by tensor



Summary of Today's Lecture

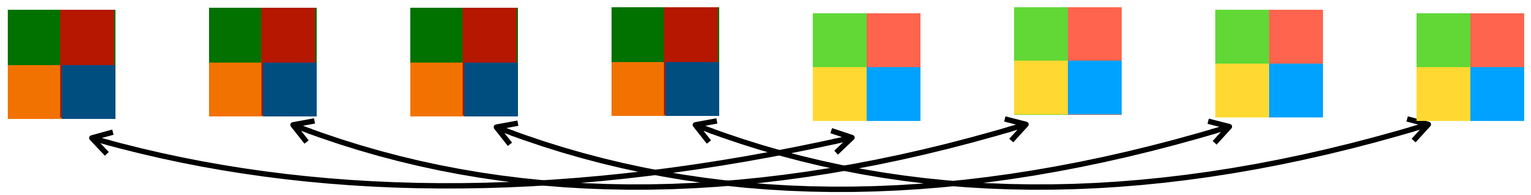
- Different parallelisms.
- Communication primitives.





Summary of Today's Lecture

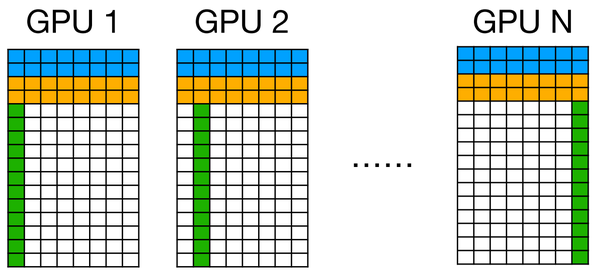
- Different parallelisms.
- Communication primitives.
  - Ring-AllReduce



Recursive Halving AllReduce

Summary of Today's Lecture

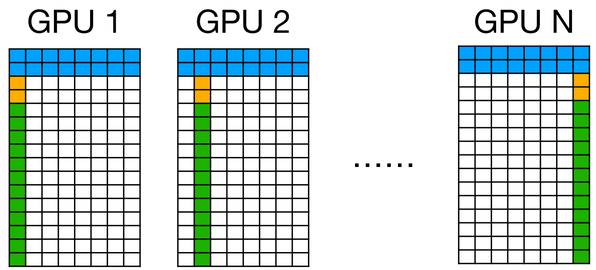
- Different parallelisms.
- Communication primitives.
  - Ring-AllReduce
- Zero-1,2,3 and FSDP



ZeRO-1

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

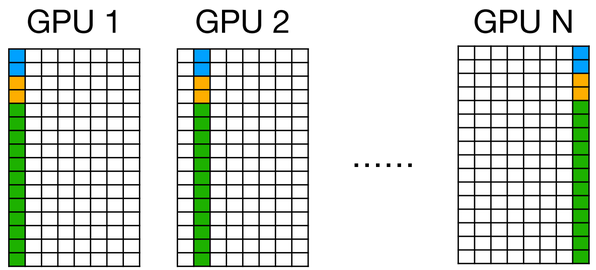
$$80\text{GB}/4.2\text{Bytes} = 19B$$



ZeRO-2

$$(\underbrace{2\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

$$80\text{GB}/2.2\text{Bytes} = 36B$$



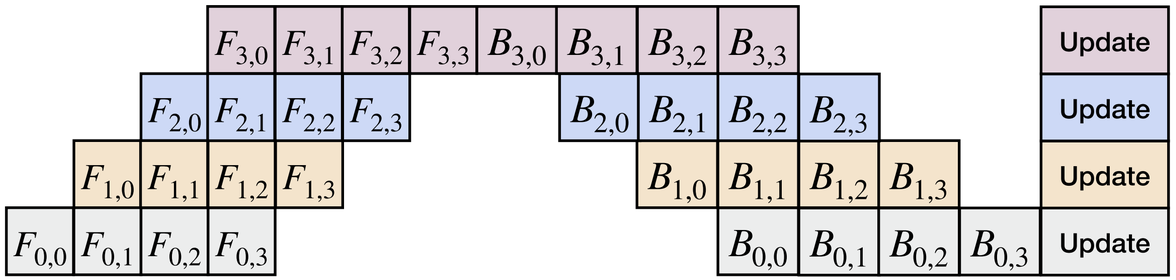
ZeRO-3

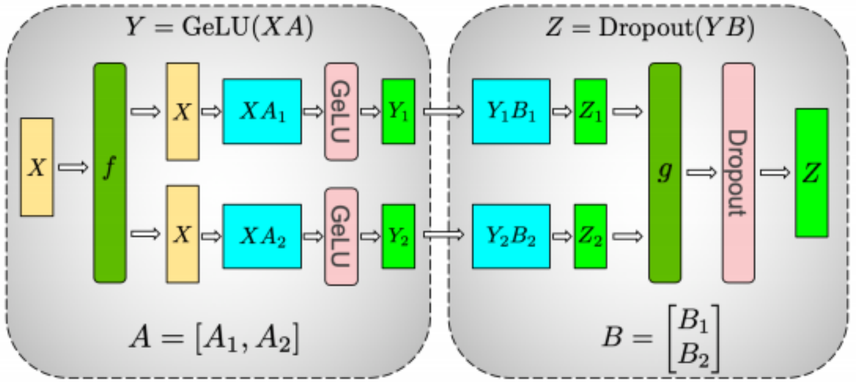
$$(\underbrace{2/N\text{bytes}}_{\text{weights}} + \underbrace{2/N\text{bytes}}_{\text{gradients}} + \underbrace{12/N\text{bytes}}_{\text{optim states}})\psi$$

$$80\text{GB}/0.25\text{Bytes} = 320B$$

Summary of Today's Lecture

- Different parallelisms.
- Communication primitives.
  - Ring-AllReduce
- Zero-1,2,3 and FSDP
- Implement pipeline parallelism and tensor parallelism.





Summary of Today's Lecture

1. Background and motivation
2. Parallelization methods for distributed training
3. Data parallelism
4. Communication primitives
5. Reducing memory in data parallelism: ZeRO-1 / 2 / 3 and FSDP
6. Pipeline parallelism
7. Tensor parallelism

References

- TSM: Temporal Shift Module for Efficient Video Understanding [Lin et al. 2019]
- Scaling Distributed Machine Learning with the Parameter Server. [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]
- Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism [Shoeybi, et al, 2019]
- ZeRO: Memory Optimizations Toward Training Trillion Parameter Models [Rajbhandari, et al, 2019]