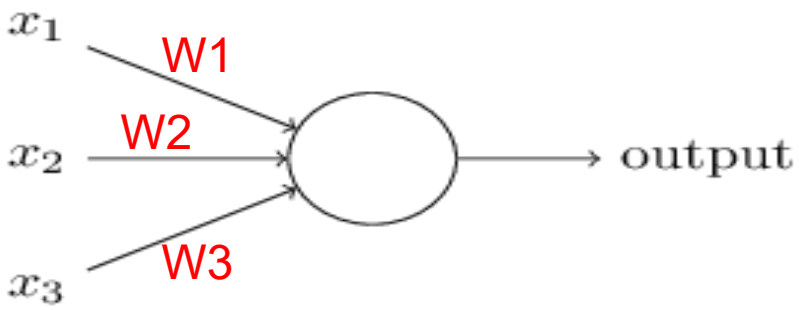
# PipeDream: Generalized Pipeline Parallelism for DNN Training

츄씬 72220689

- DNN is widely used, but the model needs to be trained before it can be deployed on the device
- Model training is time and compute intensive!
- So we need to speed up model training through parallel computing

# BACKGROUND

**perceptron model**:




- A linear relationship is learned between output and input, and intermediate output results are obtained

$$z = \sum_{i=1}^{m} w_i x_i + b$$

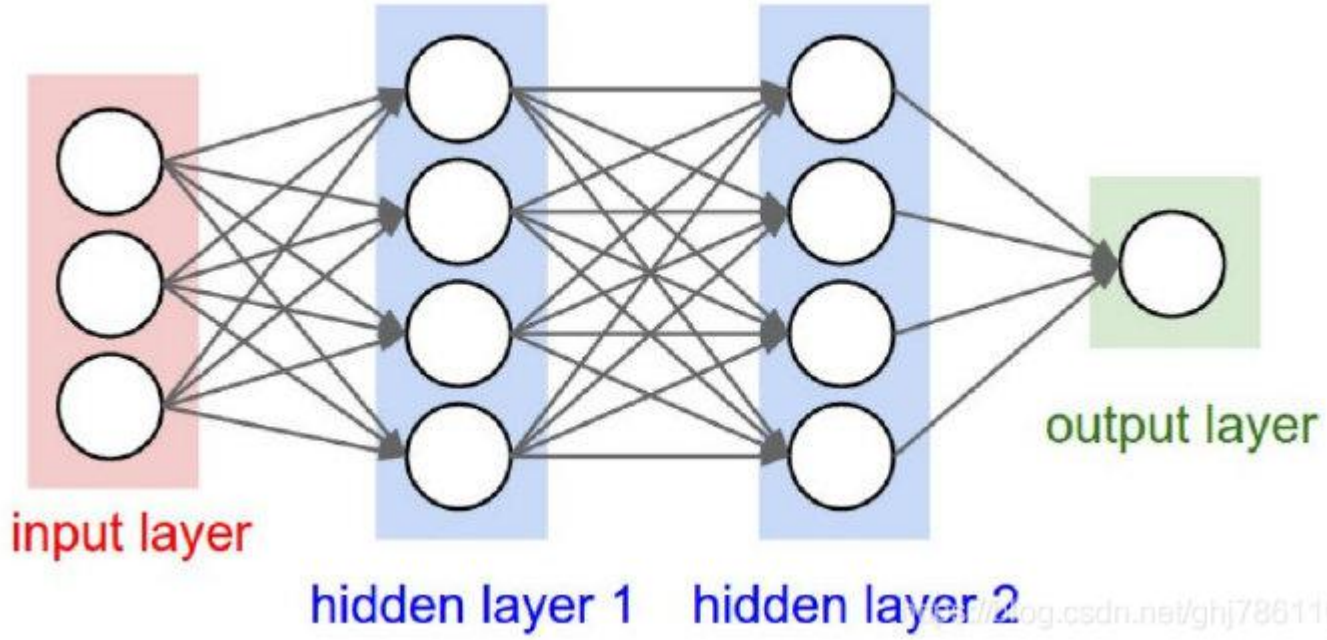
- Next is a neuron activation function (activation functions such as sign, sigmoid, )

$$sign(z) = \begin{cases} -1 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

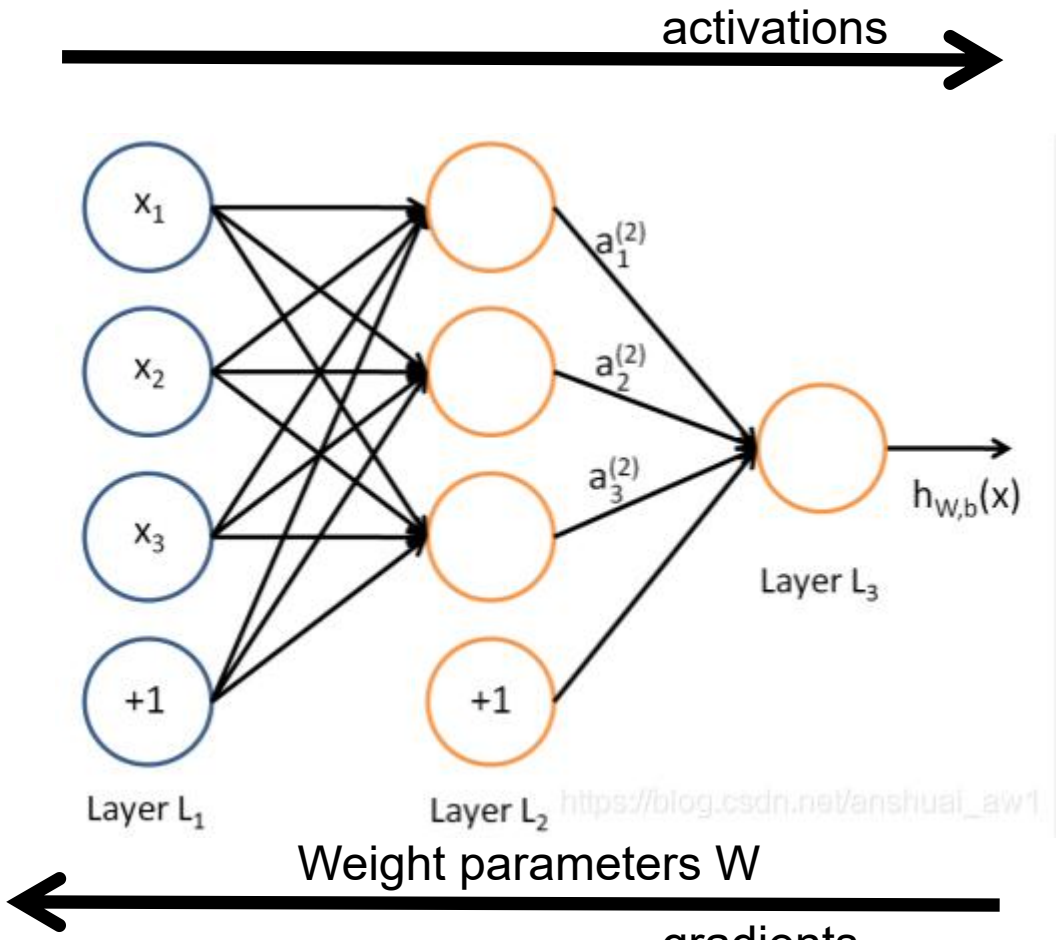
# BACKGROUND

**DNN: input layer，hidden layer and output layer**




- Use full connections between each layer

# BACKGROUND




**DNN forward propagation:**

- Use the output of the previous layer to calculate the output of the next layer

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l)$$

**DNN Backward propagation:**

- Iterative optimization of DNN loss function with gradient descent method to find the minimum value,and get W

- w optimized using standard iterative optimization procedures

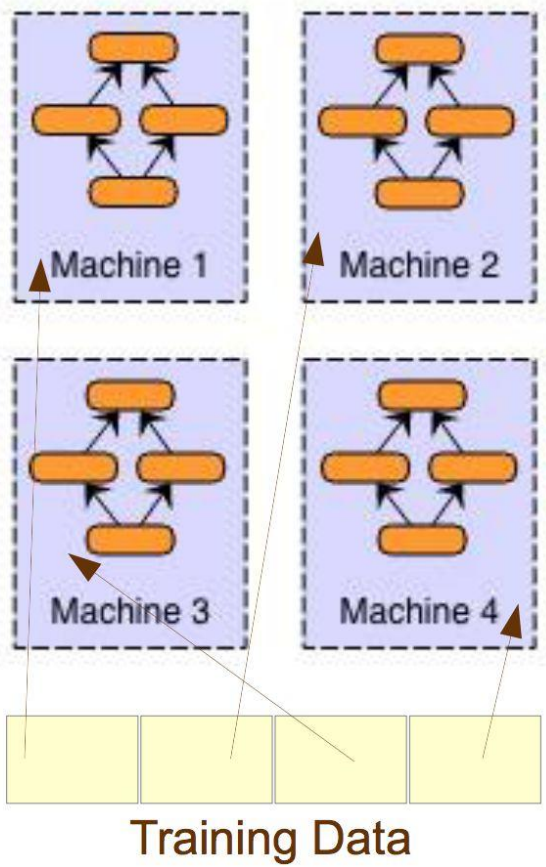
# BACKGROUND

- **intra-batch Parallelism**
  Data Parallelism
  model parallelism

- **Inter-batch Parallelism**
  GPipe

➢ Intra-batch batches mainly focuses on how to complete the training of multiple batches faster.and Inter-batch parallelism focuses on how to complete a batch of training faster

# Data Parallelism



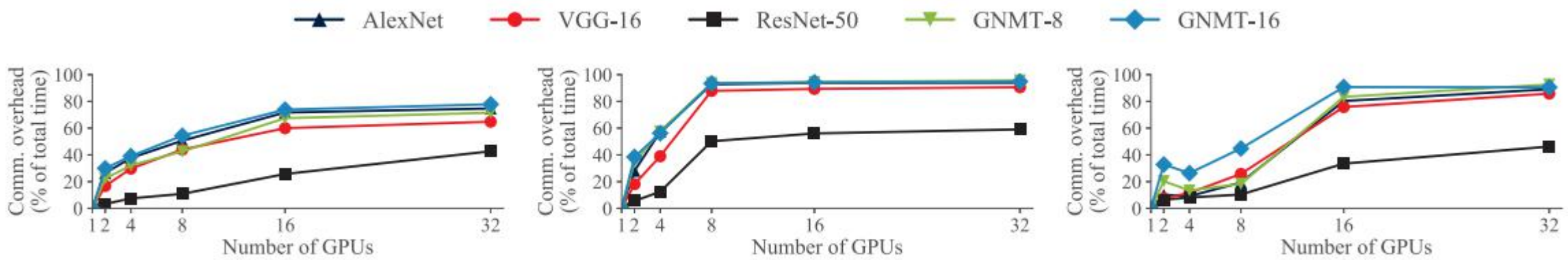

1. The input data is divided into different parts and sent to different workers

2. These computing nodes save the latest copy of the model locally

3. Model synchronization between nodes through periodic synchronization

# Data Parallelism

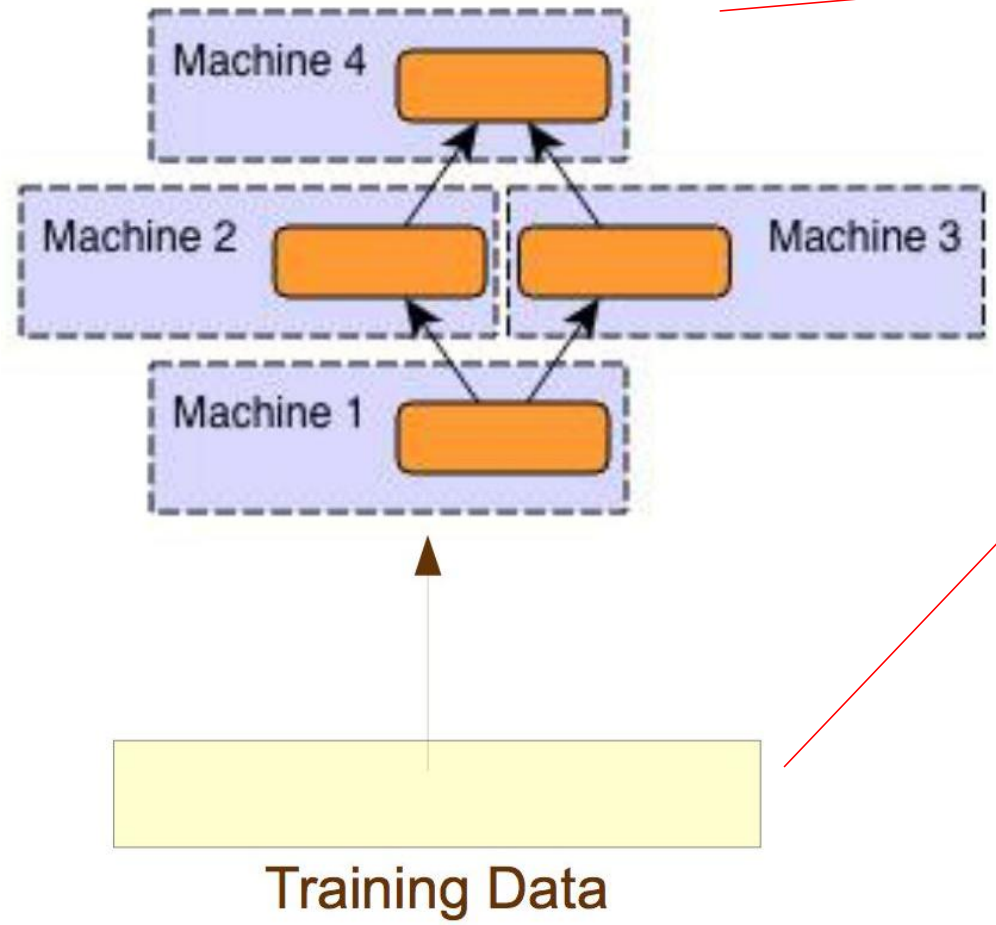
- Despite many performance optimizations, communication overhead high!




(a) Instances with 8 1080Tis (private cluster). (b) Instances with 4 V100s (Azure). (c) Instances with 8 V100s and NVLink (EC2).

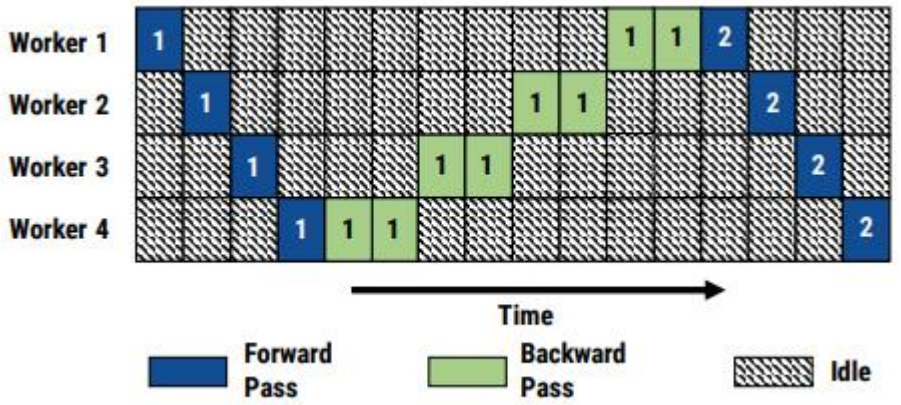
ReNet-50 has a compact weight representation, which enables it to scale well for parallel data

# Model Parallelism



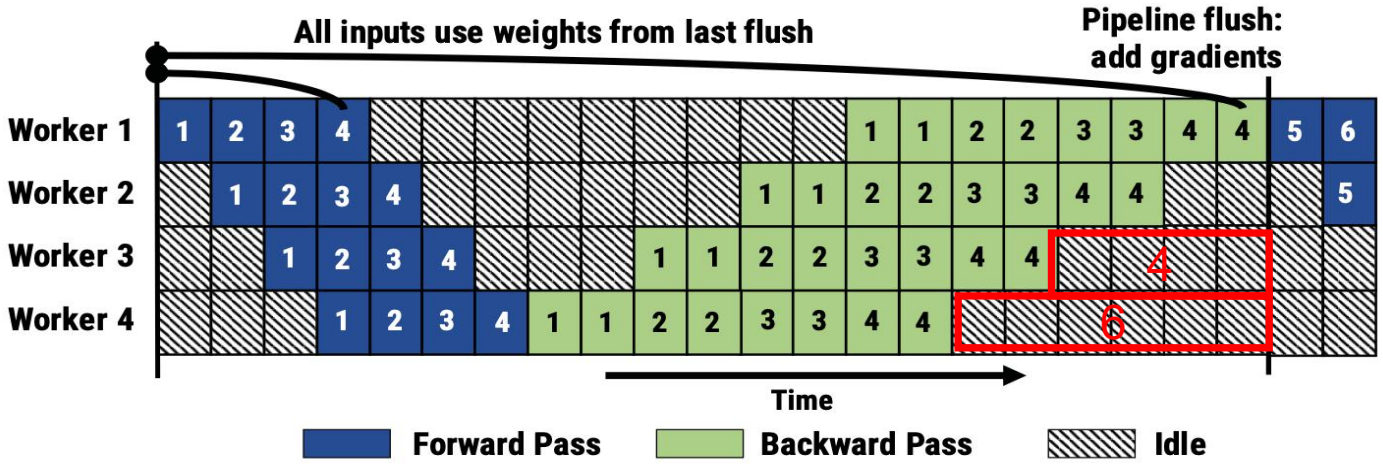

1. Divide the model into different parts to different computing nodes, each computing node is responsible for computing a part of the model

2. After each computing node completes the calculation, it needs to send the intermediate calculation result to the node responsible for the subsequent layers



# Inter-batch Parallelism:

**GPipe：Split a batch of data again into some smaller micro-batches**

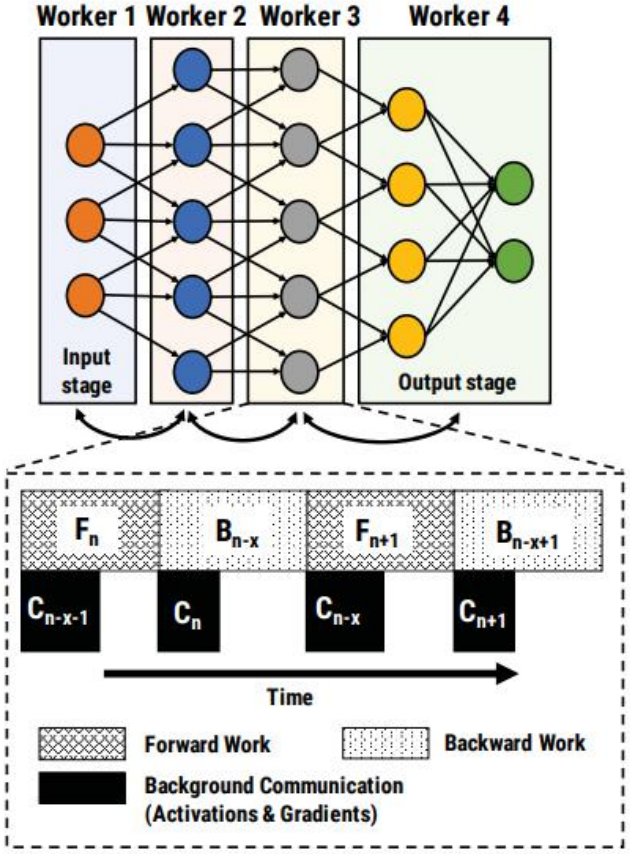



- In the figure, a batch is divided into four micro-batches, denoted by 1, 2, 3, and 4.
- The synchronous training method is also used in GPipe

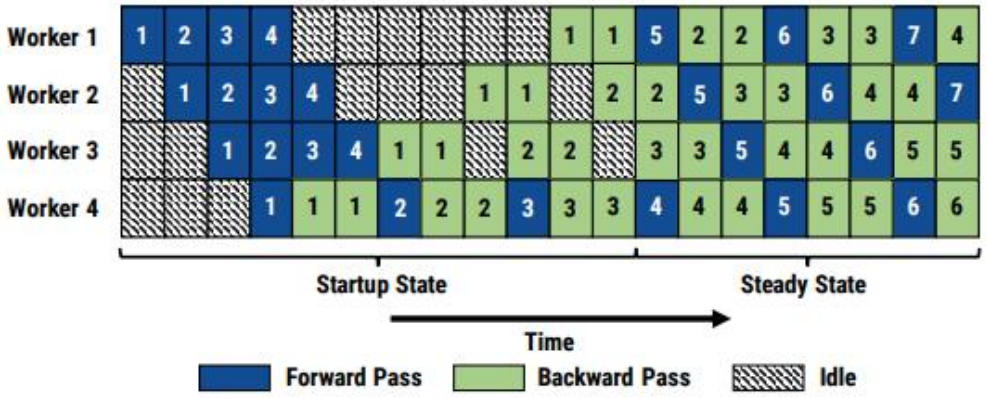
- worker 4 is idle for 6 periods after the reverse calculation process is over

# PipeDream

- **PipeDream is an approach to pipeline parallelism**

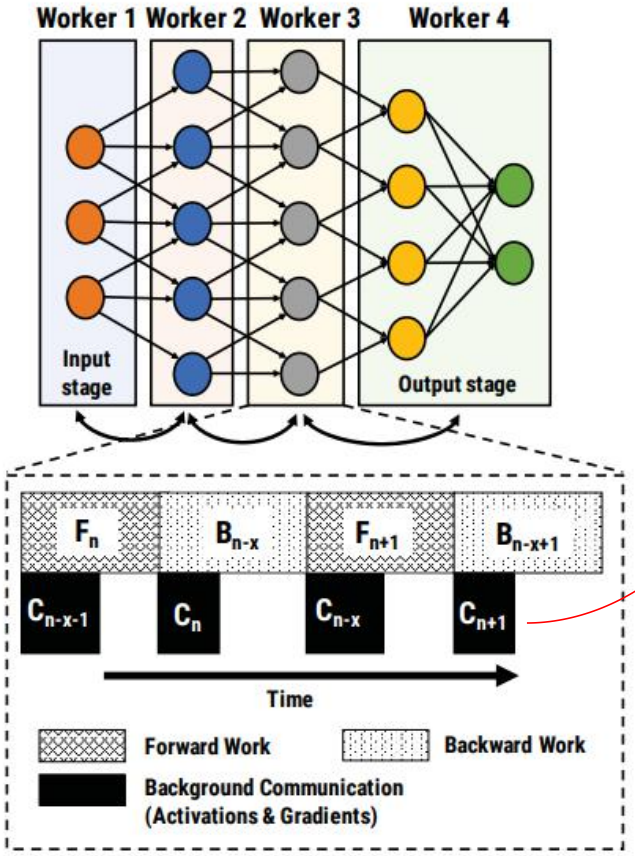



- PipeDream divides the DNN model into different stages, each stage can contain one or more layers, and different computing nodes (mainly a GPU here) calculate different stages (including forward calculation and reverse calculation) respectively. to calculate)



# PipeDream

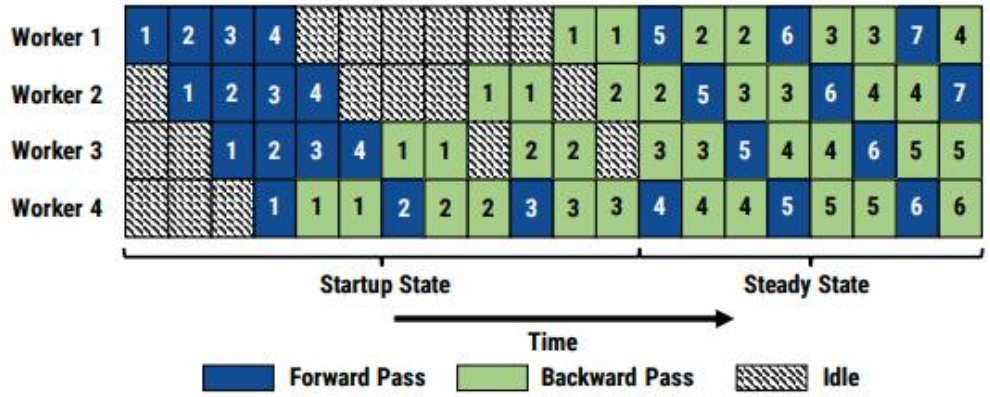
- **PipeDream is an approach to pipeline parallelism**



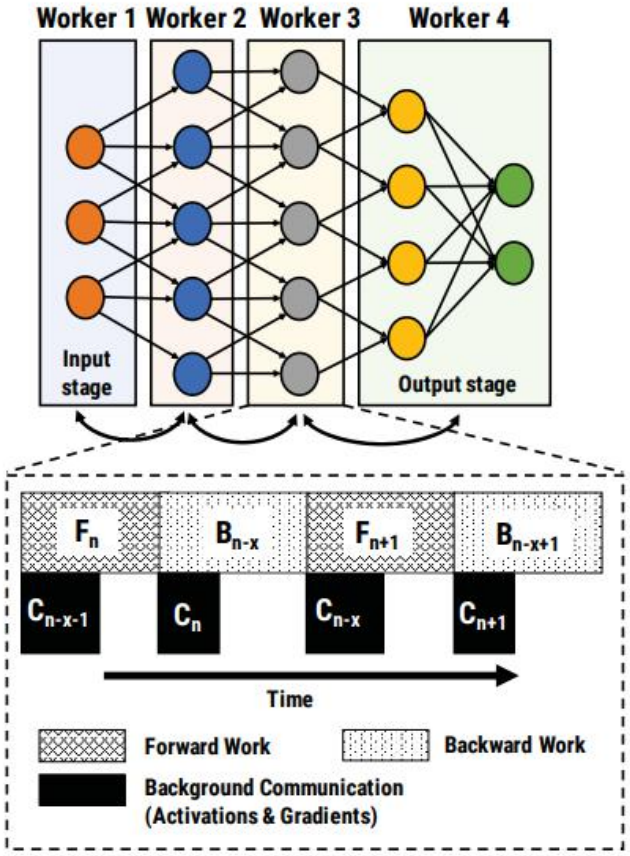

➢ Differences from GPipe

1. After getting the gradient of a certain stage (layer), update it immediately(asynchronous)

2. Communication and computation are parallel



# PipeDream

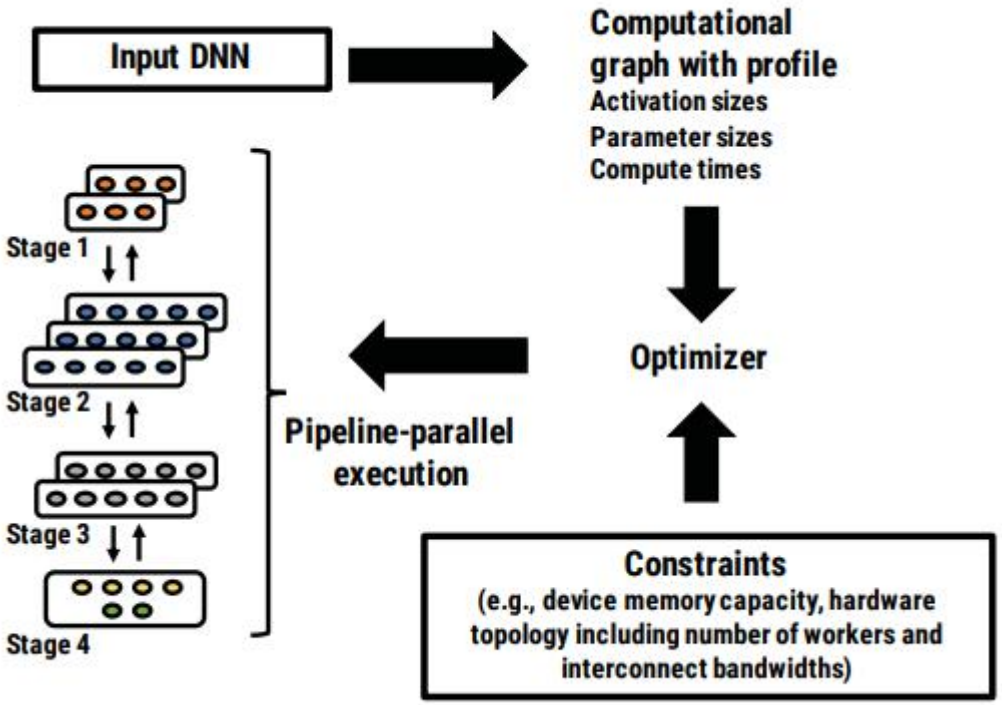



- **Advantages of PipeDream**

- PipeDream's pipeline communication is greatly reduced
- The computation and communication in PipeDream overlap in time,this greatly improves efficiency

Pipeline-parallel training up to **5.3x faster** than data parallelism without sacrificing on final accuracy of the model

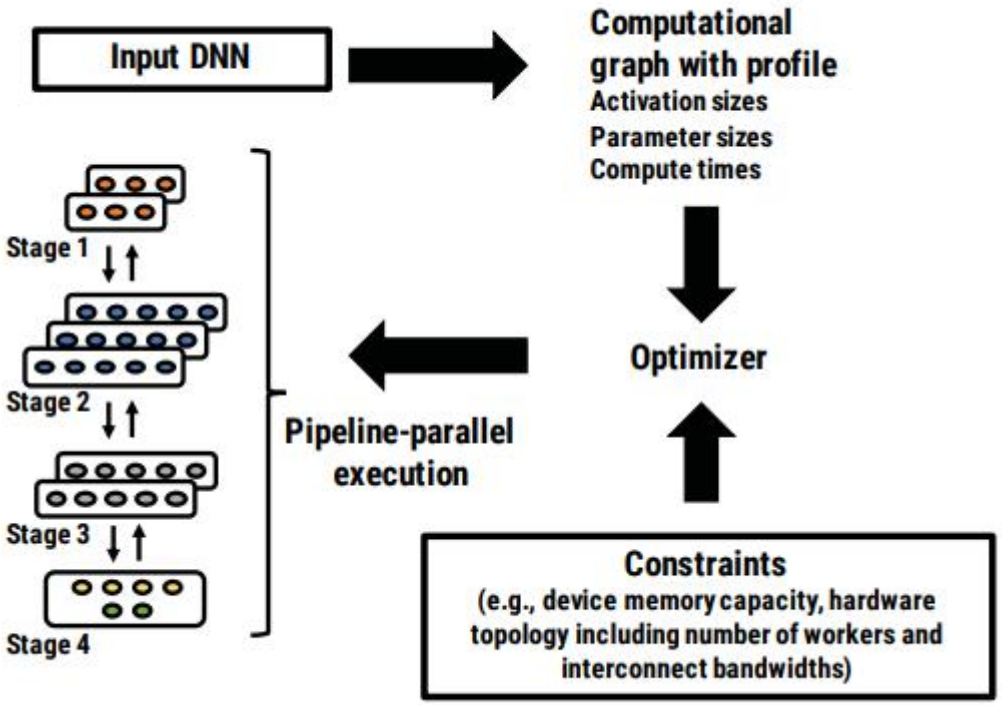
# PipeDream




## PipeDream Workflow

1. Using the structure of the DNN model as input, get the basic parameter information of the model,(For example, the size of the parameters of each layer, the calculation time, the size of the activation value, etc.)
2. According to the result calculated in the first step, combined with other information,Calculate an optimal model segmentation method (the model is divided into multiple stages)
3. Use PipeDream runtime (runtime) combined with input data to train the DNN model after segmentation optimization

# PipeDream:




- **Problems to be solved in PipeDream**

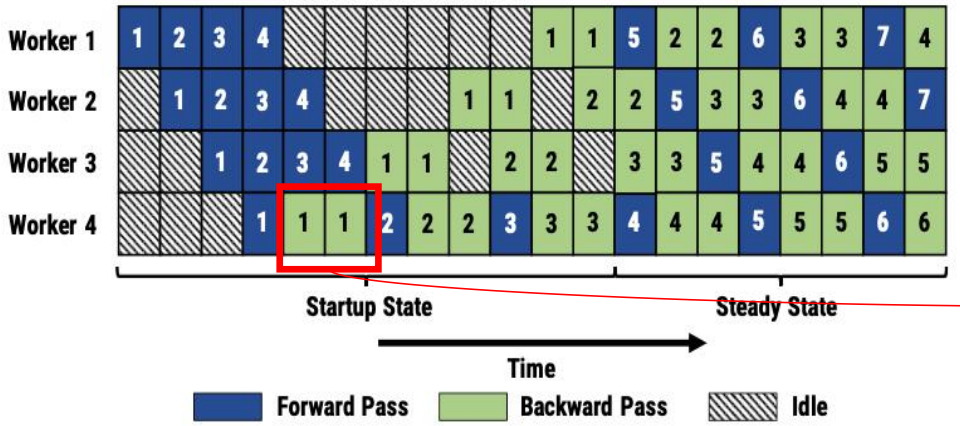
1 How  should the operators in a DNN model be patitioned into pipeline stages?

2 How should forward and backward passes of differnt inputs be scheduled?

3 How should weight and activation versions be managed?

# PipeDream-Work Scheduling

- The paper proposes a very simple and practical scheduling scheme - **1F1B(one forward one backward)**



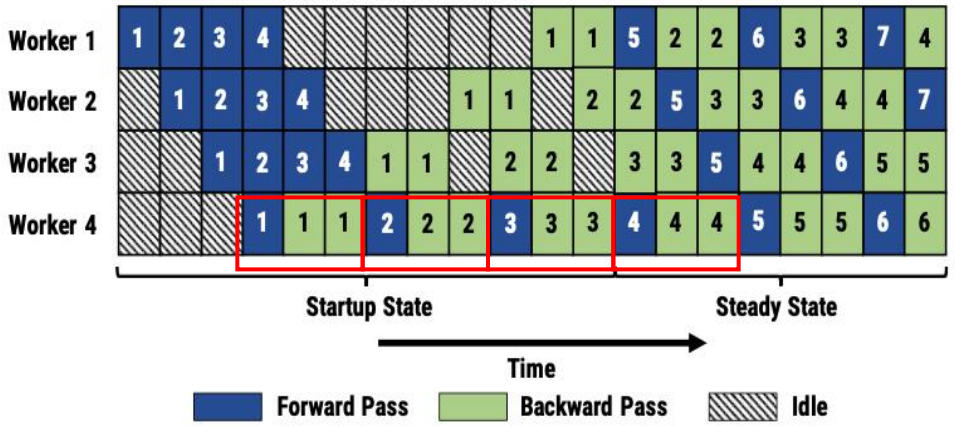

**1F1B：** the pipeline in the steady state. A backpropagation calculation is performed every time a forward calculation is performed

➢ The difference from Gpipe pipeline is:

1. Backpropagation computations with lower numbers are inserted before forward computations in subsequent micro-batches

# PipeDream-Work Scheduling

- The paper proposes a very simple and practical scheduling scheme - **1F1B(one forward one backward)**

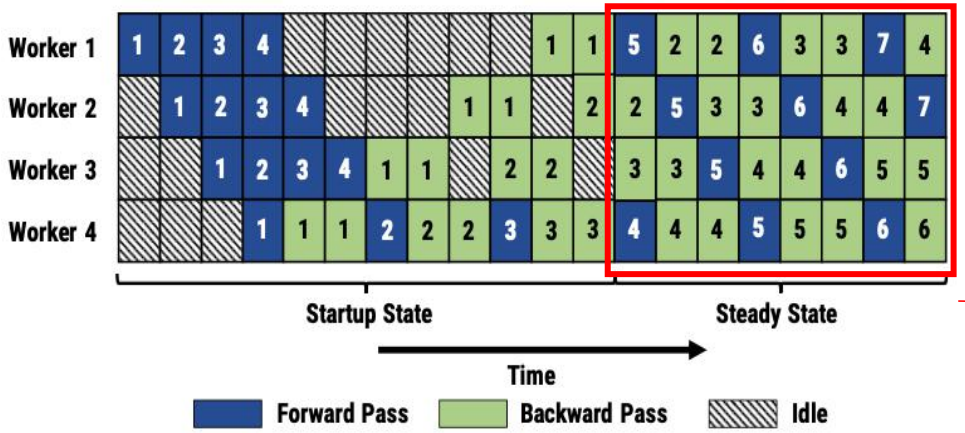



➢ The difference from Gpipe pipeline is:

2 After a batch calculation is completed, it no longer waits for other micro-batch calculations to complete and synchronizes all calculation results, but directly updates the model after backpropagation calculates a gradient (asynchronous)

# PipeDream-Work Scheduling

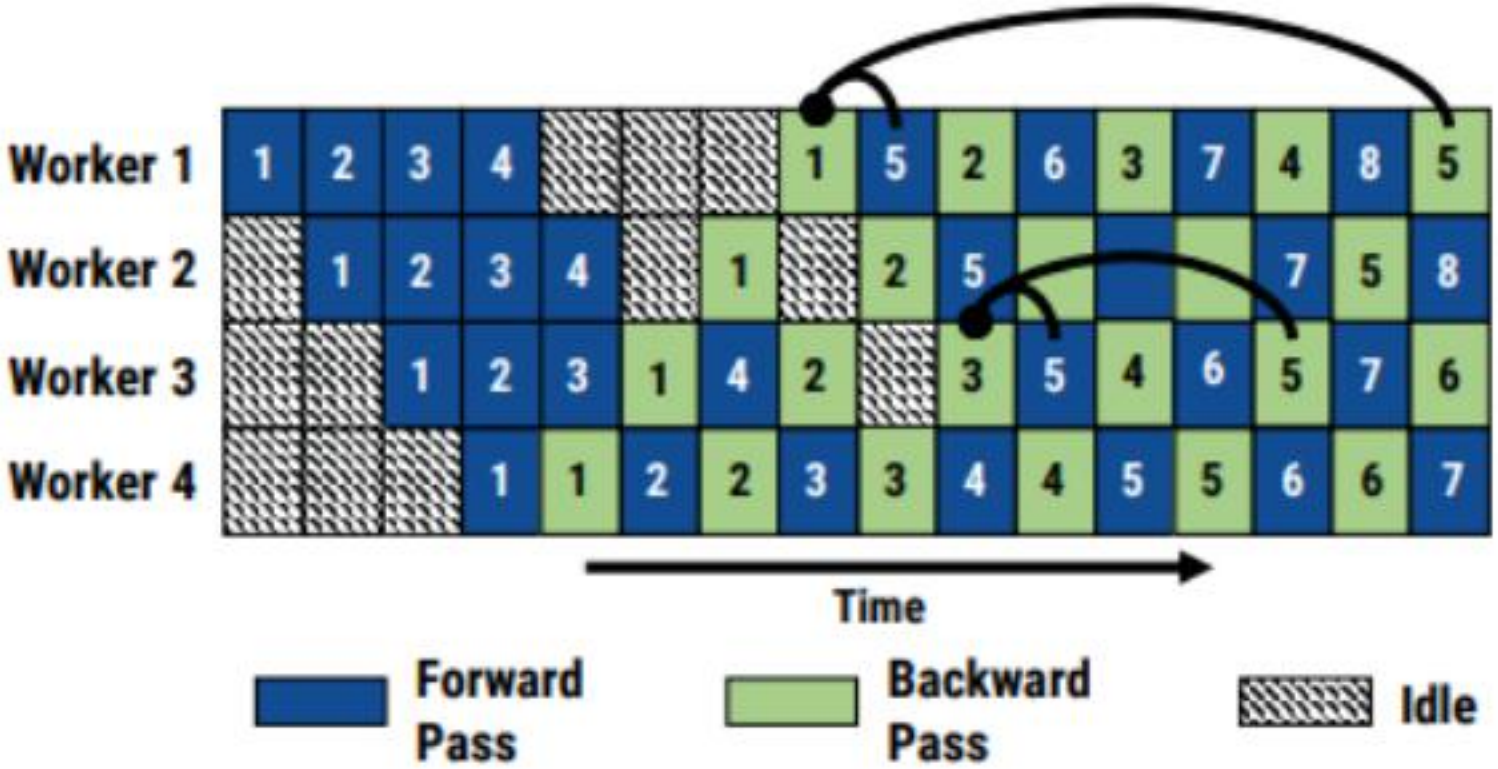
- The paper proposes a very simple and practical scheduling scheme - **1F1B(one forward one backward)**



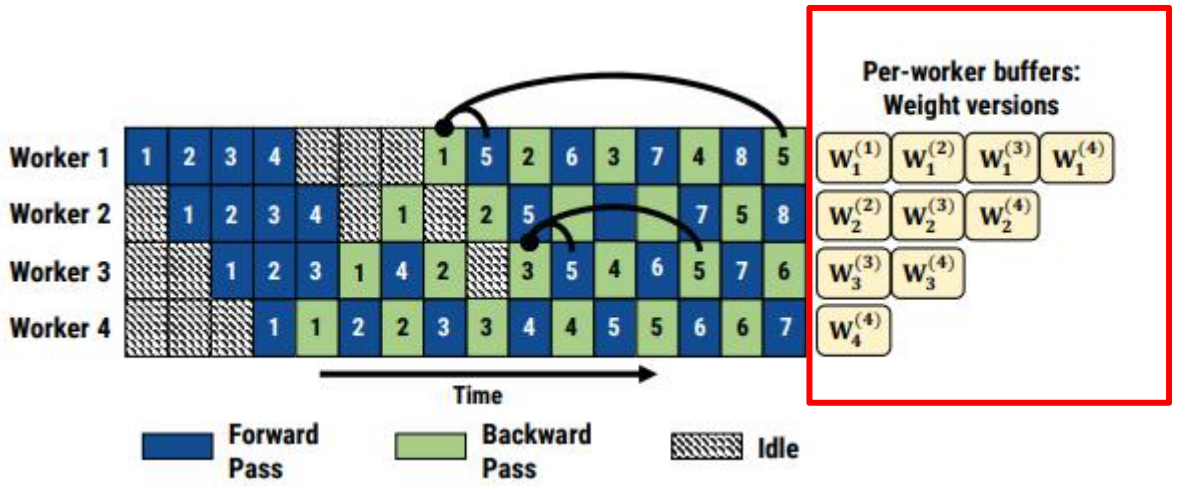

- **It can be clearly seen that the utilization of computing resources is improved**
- **In the steady state of the pipeline, the theoretical computing resource utilization rate reaches 100%**

# PipeDream- Effective Learning

**Problem:** PipeDream uses asynchronous calculation. If no processing is done, it will cause some batches to use different model parameters during forward and backward propagation



# PipeDream- Effective Learning




- **Weight stashing**

➢ Propose to cache the previous version of the model before each update

➢ Versioning these cached models

Store **multiple<weight,activation> versions**

# Evaluation Setup

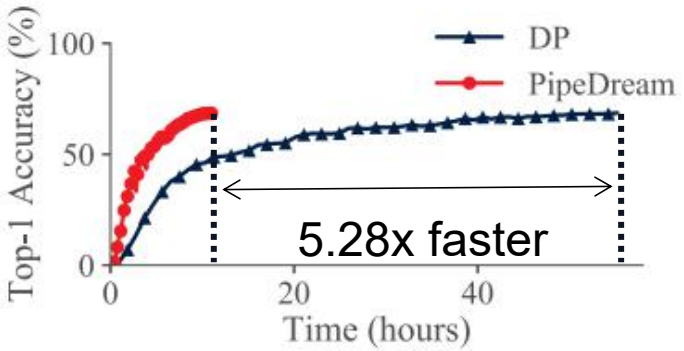
- Integrated PipeDream with PyTorch in ~3000 lines of Python code
- Integrated with PyTorch's communication library
  - NCCL backend for Data Parallelism baselines
  - Gloo backend for PipeDream
- Experiments run on three different server types
  - Cluster A: 4xV100 GPUS,PCIe intra-server,and 10Gbps inter-server (Azure)
  - Cluster B: 8xV100 GPUs, NVLink intra-server,and 25Gbps inter-server(AWS)
  - Cluster C: 1xTian X, and 40 Gbps inter-server(private)

# Evaluation Setup

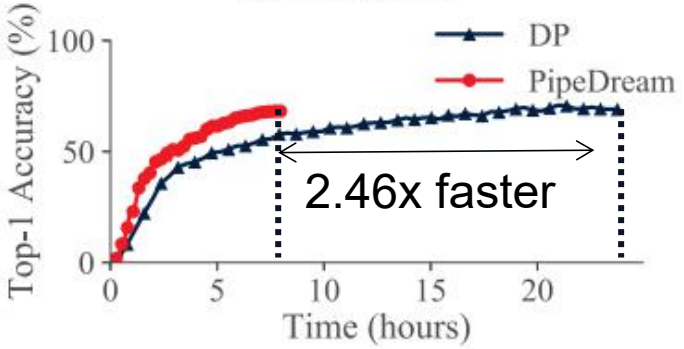
| Task | Model | Dataset | Accuracy Threshold | # Servers × # GPUs per server (Cluster) | PipeDream Config | Speedup over DP | |
|---|---|---|---|---|---|---|---|
| | | | | | | Epoch time | TTA |
| Image Classification | VGG-16 [48] | ImageNet [44] | 68% top-1 | 4x4 (A) | 15-1 | 5.28× | 5.28× |
| | | | | 2x8 (B) | 15-1 | 2.98× | 2.46× |
| | ResNet-50 [26] | ImageNet [44] | 75.9% top-1 | 4x4 (A) | 16 | 1× | 1× |
| | | | | 2x8 (B) | 16 | 1× | 1× |
| | AlexNet [37] | Synthetic Data | N/A | 4x4 (A) | 15-1 | 4.92× | N/A |
| | | | | 2x8 (B) | 15-1 | 2.04× | N/A |
| Translation | GNMT-16 [55] | WMT16 EN-De | 21.8 BLEU | 1x4 (A) | Straight | 1.46× | 2.2× |
| | | | | 4x4 (A) | Straight | 2.34× | 2.92× |
| | | | | 2x8 (B) | Straight | 3.14× | 3.14× |
| | GNMT-8 [55] | WMT16 EN-De | 21.8 BLEU | 1x4 (A) | Straight | 1.5× | 1.5× |
| | | | | 3x4 (A) | Straight | 2.95× | 2.95× |
| | | | | 2x8 (B) | 16 | 1× | 1× |
| Language Model | AWD LM [40] | Penn Treebank [41] | 98 perplexity | 1x4 (A) | Straight | 4.25× | 4.25× |
| Video Captioning | S2VT [54] | MSVD [11] | 0.294 METEOR | 4x1 (C) | 2-1-1 | 3.01× | 3.01× |

Table 1: Summary of results comparing PipeDream with data parallelism (DP) when training models to advertised final accuracy. A PipeDream config of "2-1-1" means the model is split into three stages with the first stage replicated across 2 workers, and a "straight" configuration is a pipeline with no replicated stages—e.g., "1-1-1-1" on 4 workers. Batch sizes used to train these models are reported in § 5.1.

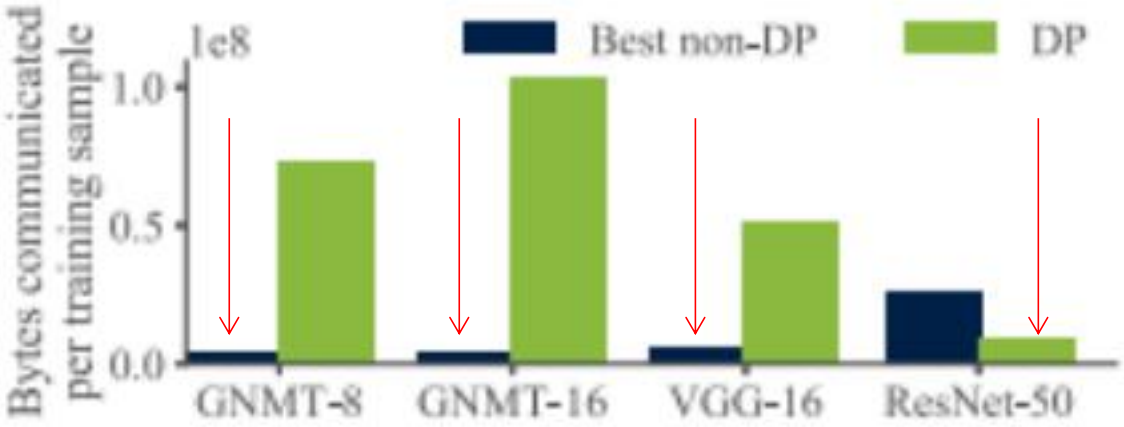
- **Accuracy vs. time for VGG-16 using 16 GPUs.**




(a) Cluster-A.




- Each circle or triangle represents two epochs of training.




- For many models , intermediate activations and gradients order of magnitude smaller than communication with Data Parallelism(DP)

# Conclusion

- Model and data parallelism often suffer from high communication overhead and low resource utilization for certain models and deployments
- PipeDream shows pipelining can be used to accelerate DNN training
- Pipelining , when combined with data and model parallelism in a principled way , achieves end-to-end speedups of up to 5.3x