CS 498: Machine Learning System Spring 2025

Minjia Zhang

The Grainger College of Engineering

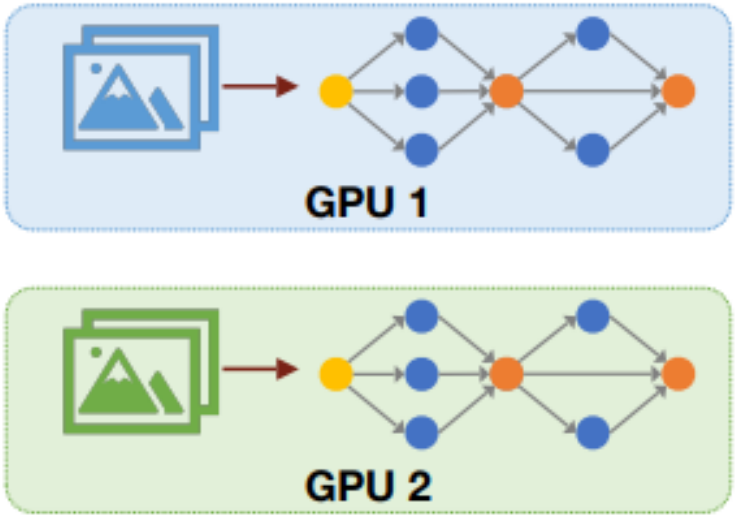
Today

Pipeline Parallelism

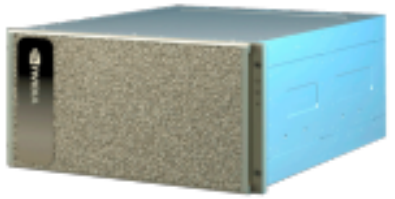
Multi-Dimensional Parallelism

First assignment (due in two weeks Mar 25 EOD)

Data Parallelism Cannot Train Large Models



175B * 16 Bits
= 350GB Memory

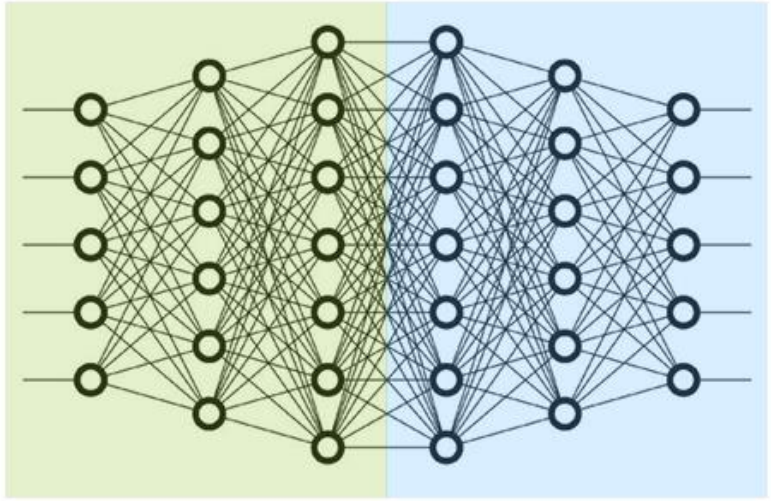


Nvidia H100 94 GB

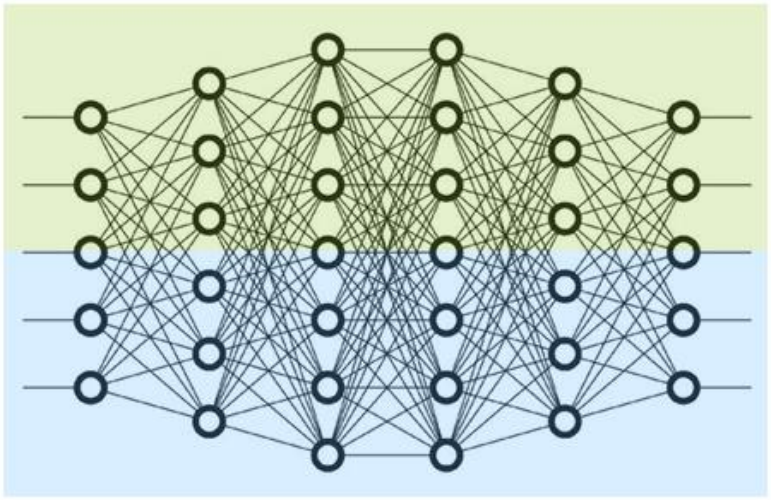
Even the best GPU **CANNOT** fit the model into memory!

Model Parallelism

Inter-layer (Pipeline) parallelism
- Split sets of layers across multiple devices
- Layer 0, 1, 2 and layer 3, 4, 5 are on different devices

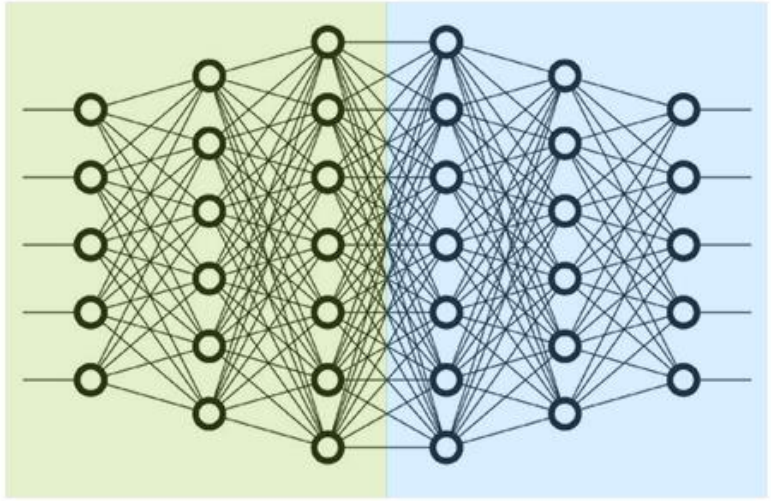


Intra-layer (Tensor) parallelism
- Split individual layers across multiple devices
- Both devices compute different parts of layer 0, 1, 2, 3, 4, 5

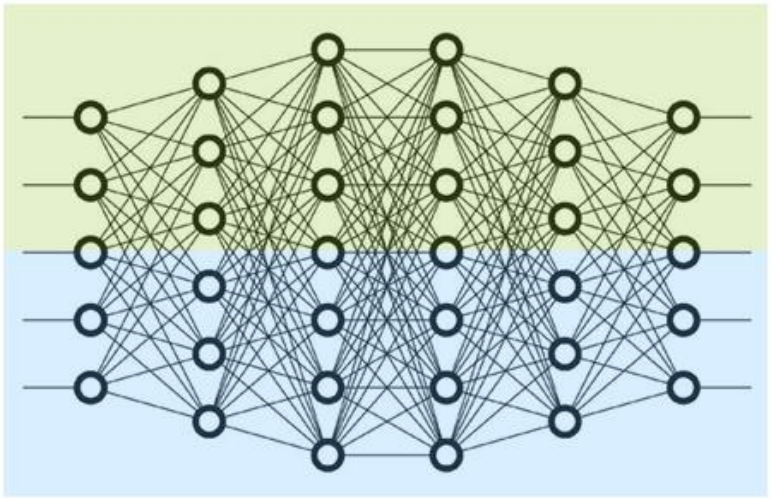


Model Parallelism

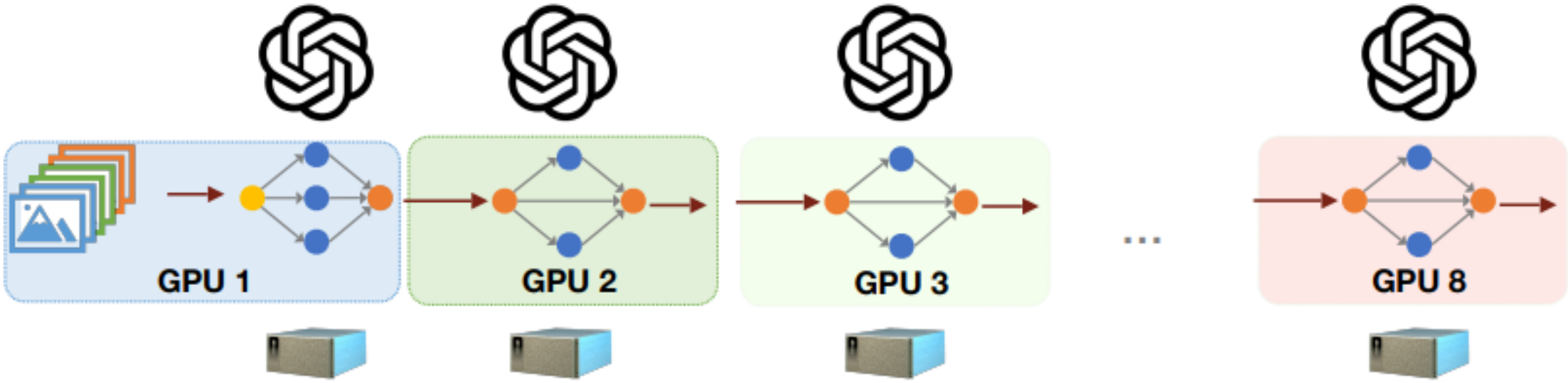
Inter-layer (Pipeline) parallelism
- Split sets of layers across multiple devices
- Layer 0, 1, 2 and layer 3, 4, 5 are on different devices



Intra-layer (Tensor) parallelism
- Split individual layers across multiple devices
- Both devices compute different parts of layer 0, 1, 2, 3, 4, 5



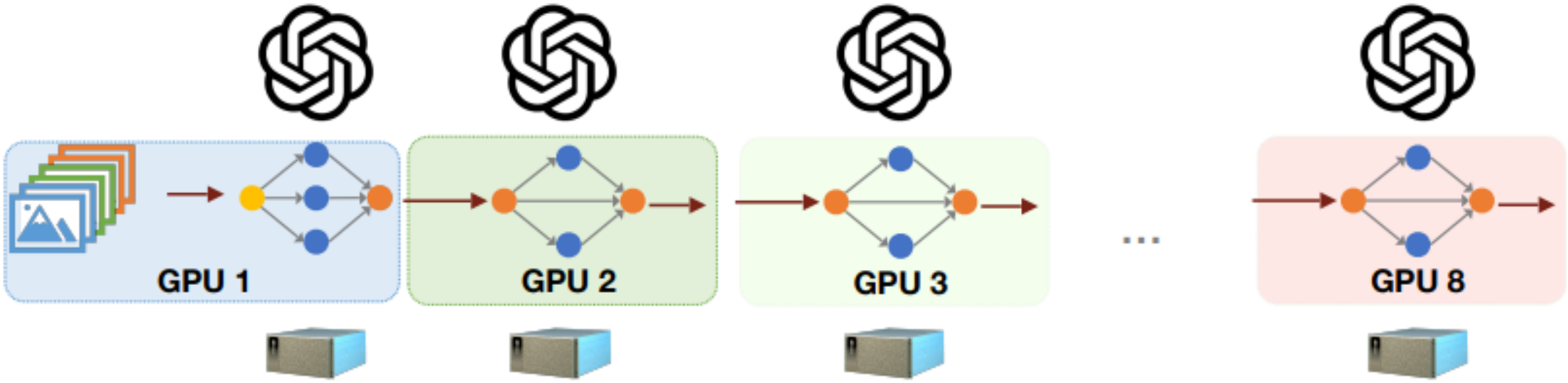
Inter-Layer Model Parallelism



350GB / 8 cards = 43.75G < 80G

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

Inter-Layer Model Parallelism

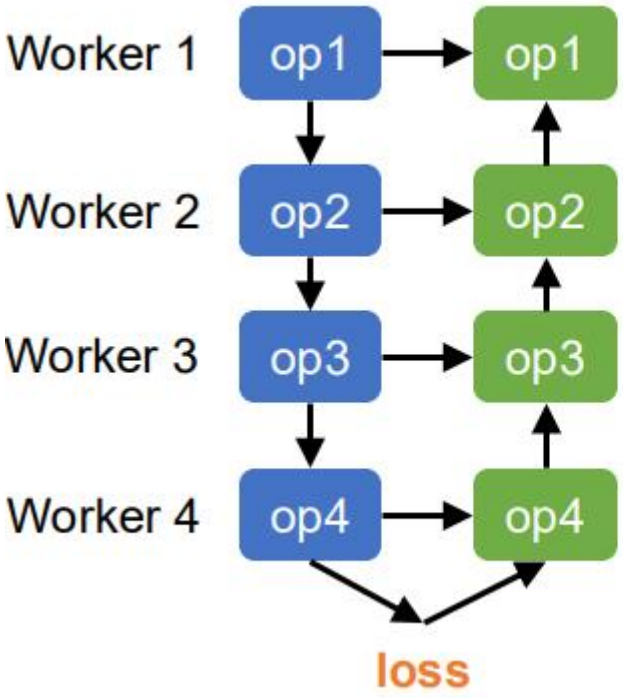


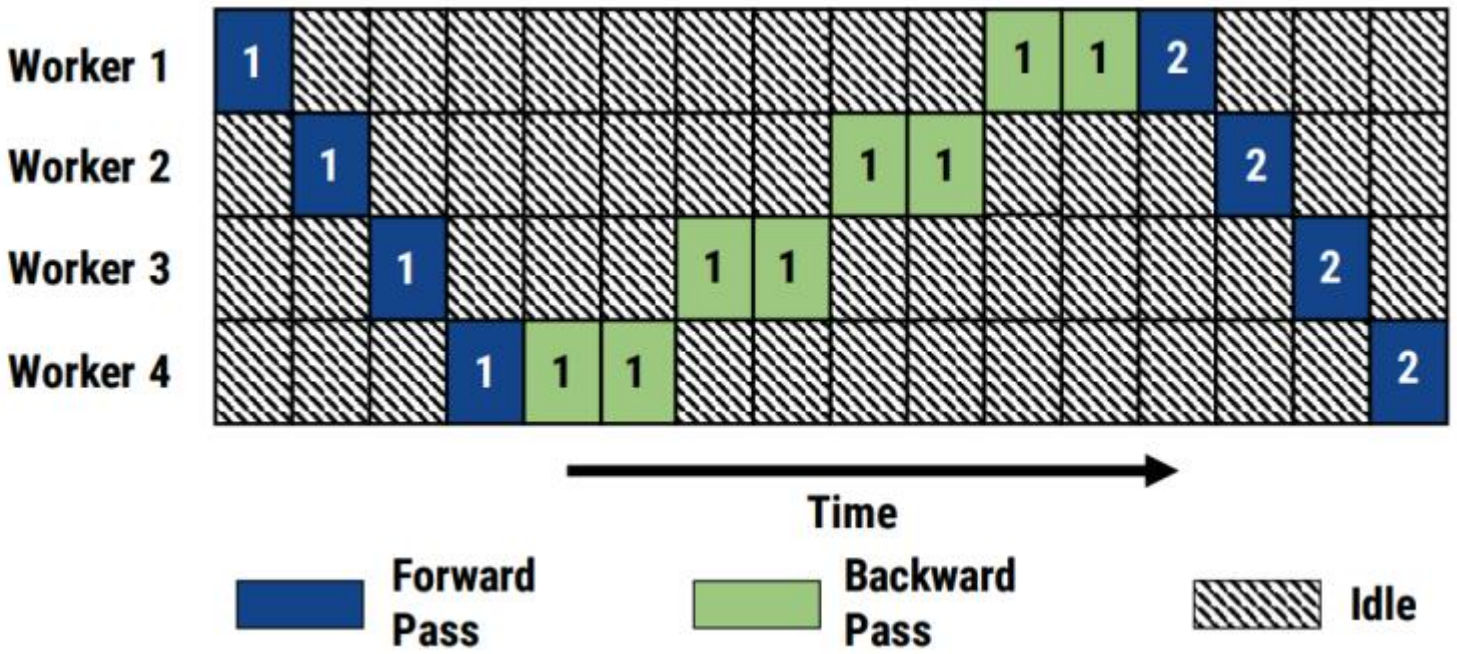
350GB / 8 cards = 43.75G < 80G

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

Question: How to achieve high training throughput through inter-layer model parallelism?

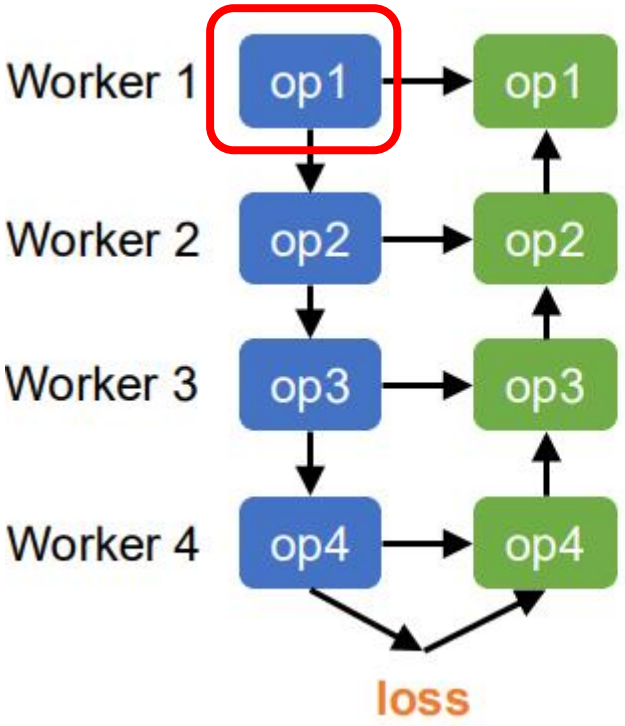
Simple Inter-Layer Parallelism

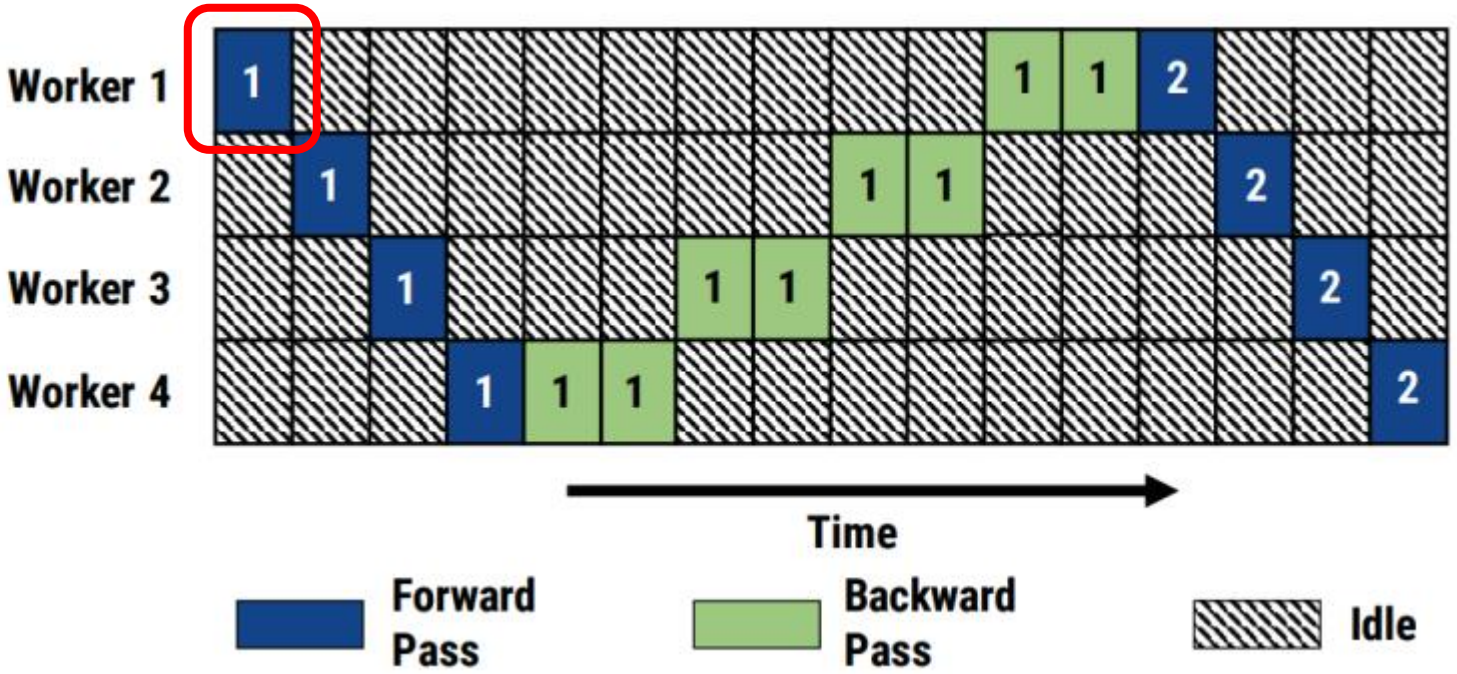




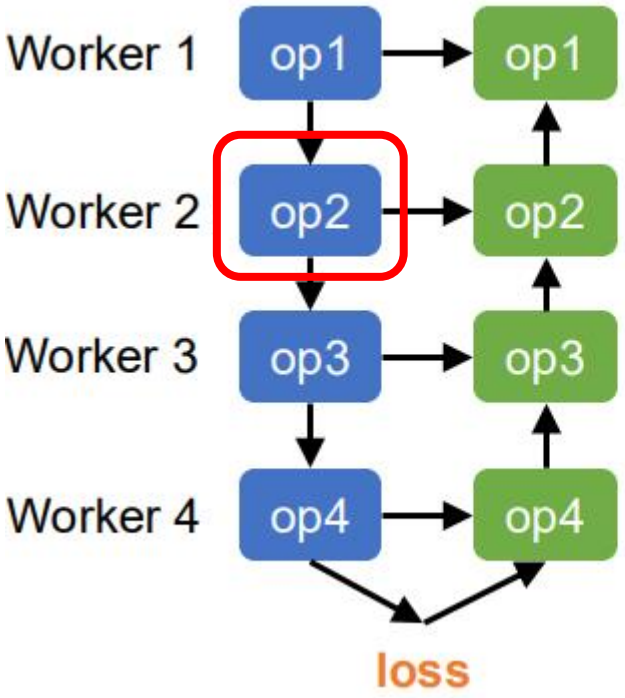
DNN training involves a bi-directional execution
- The forward pass for a minibatch starts at the input layer
- The backward pass ends at the input layer

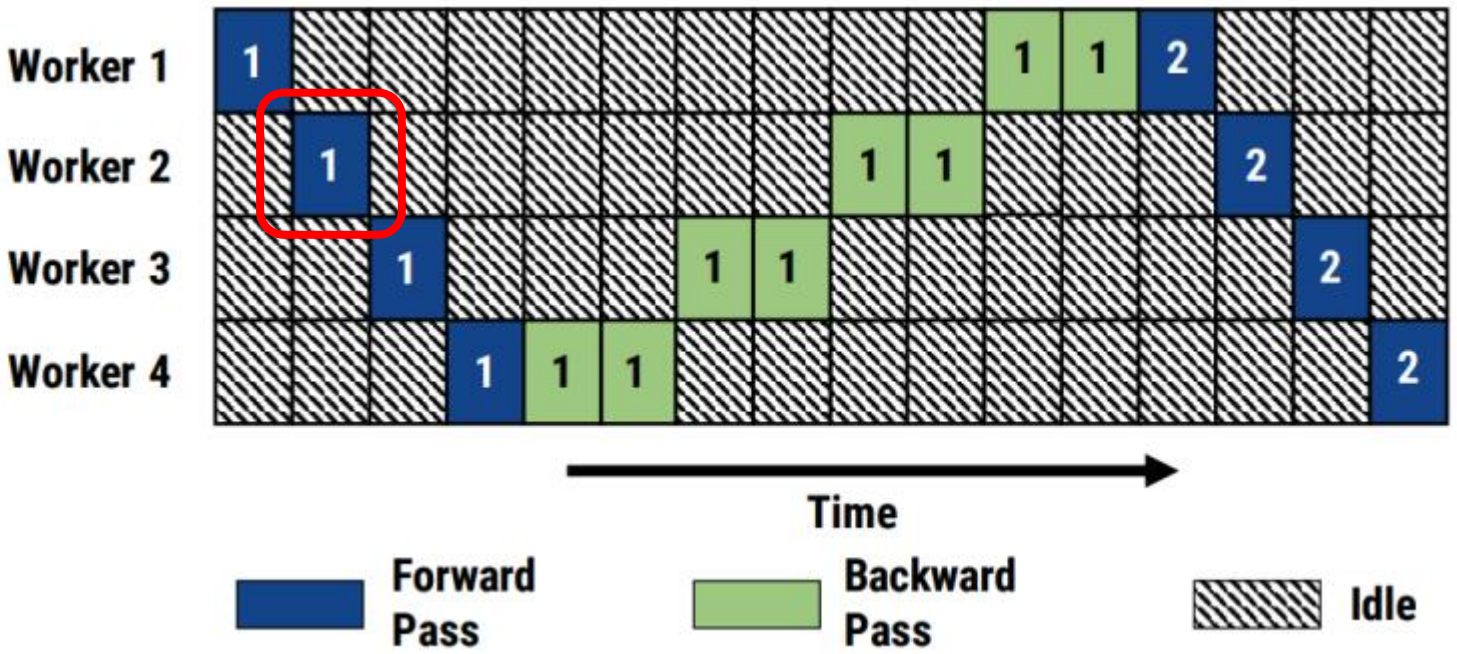
Simple Inter-Layer Parallelism



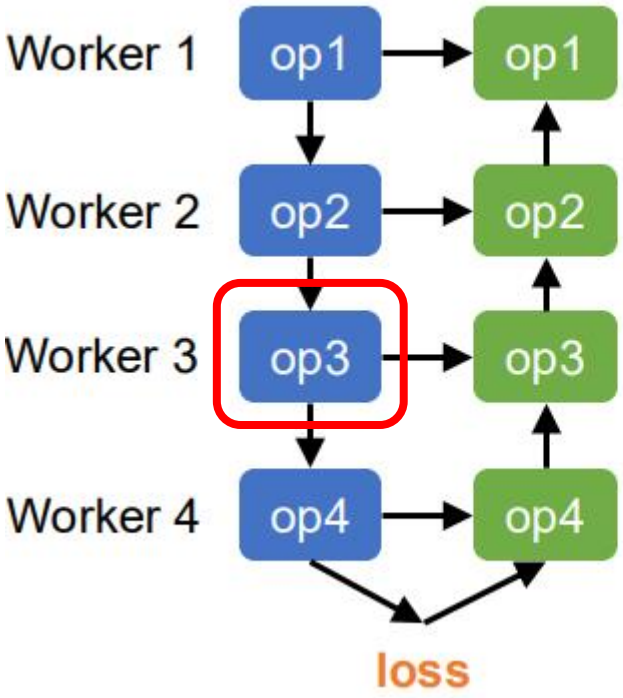


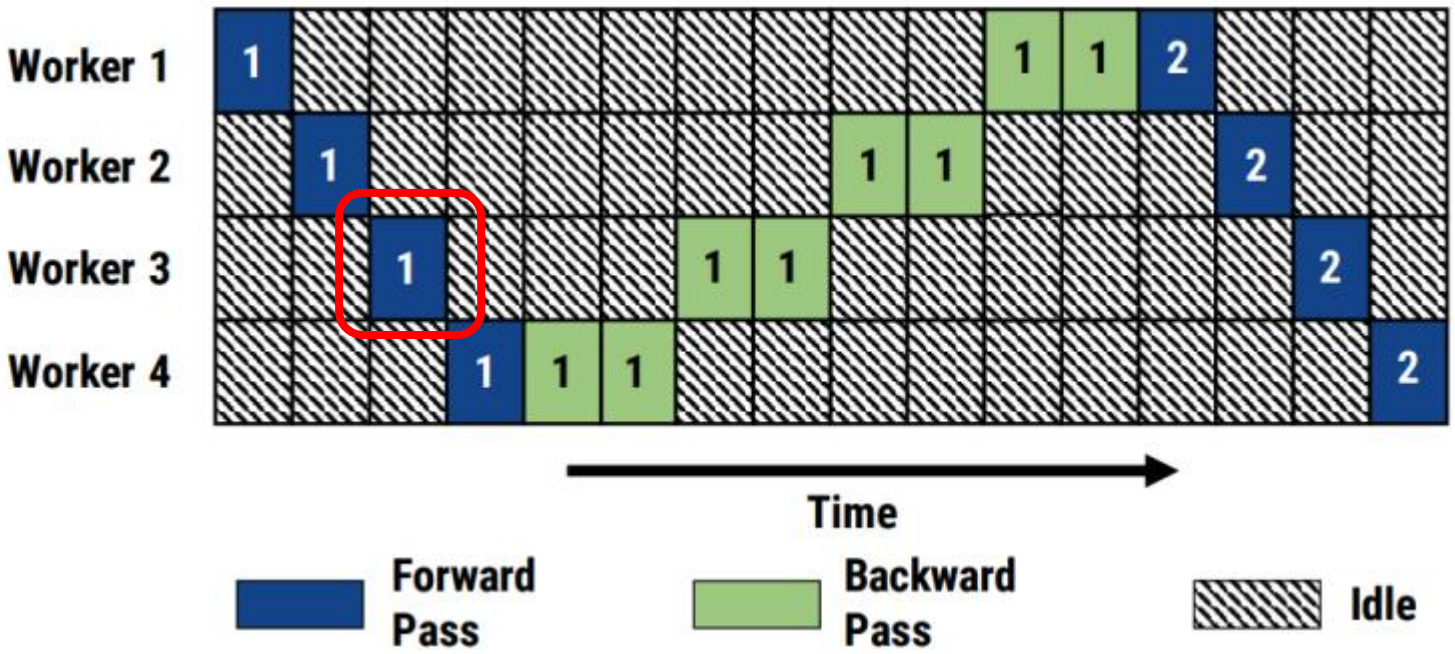
Simple Inter-Layer Parallelism



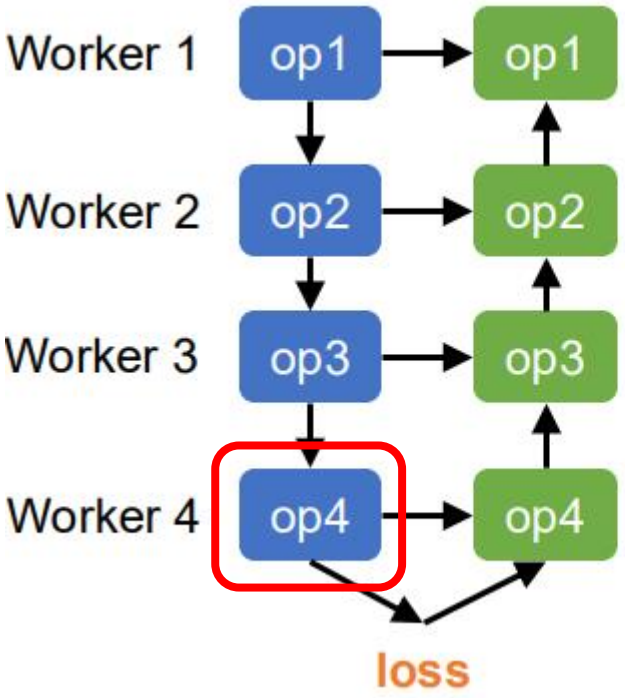


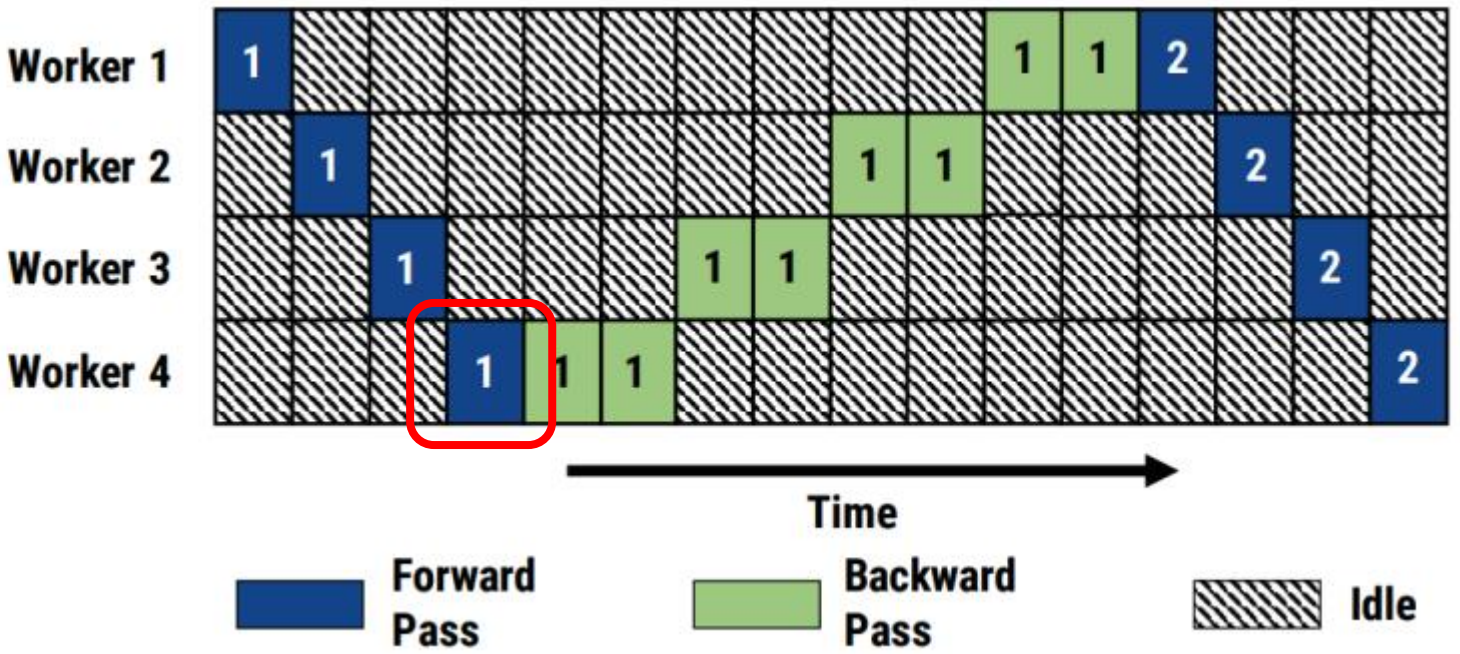
Simple Inter-Layer Parallelism



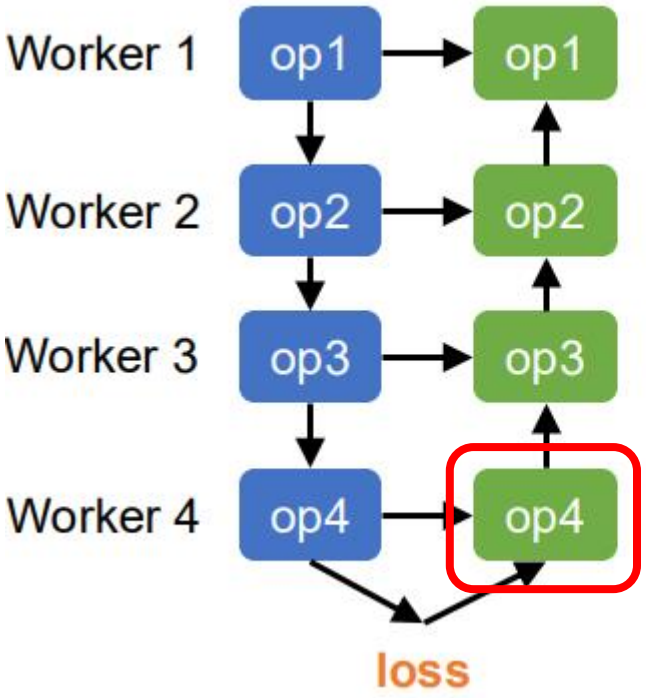


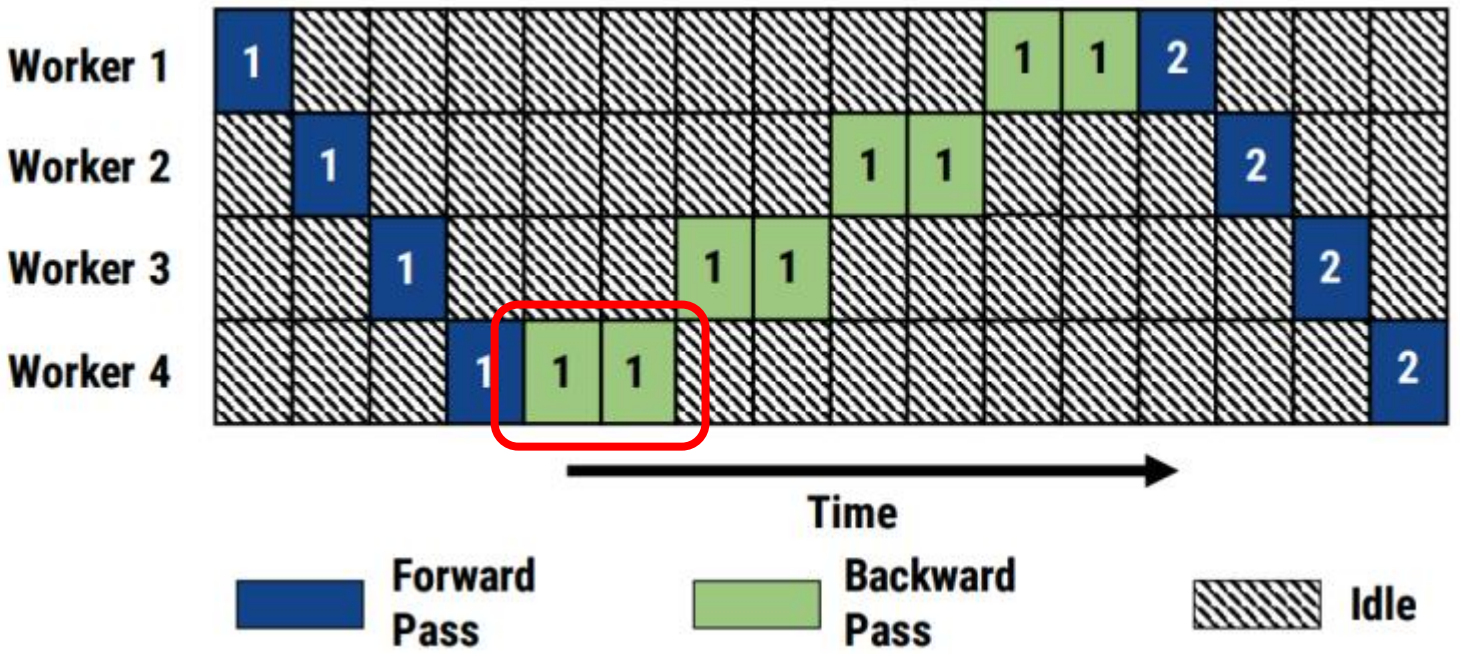
Simple Inter-Layer Parallelism



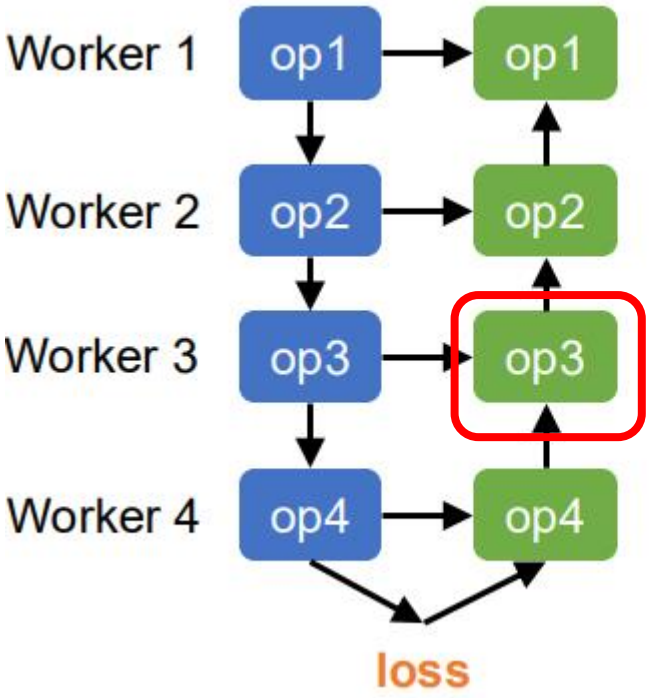


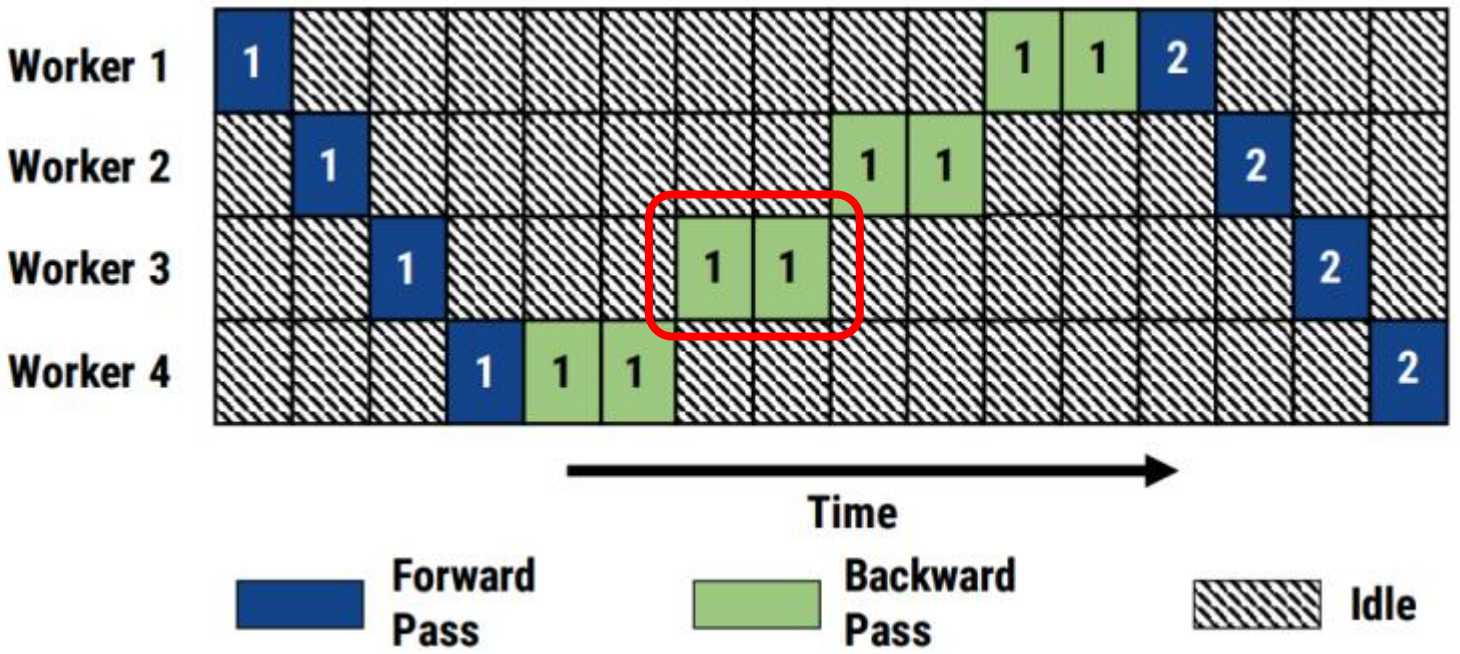
Simple Inter-Layer Parallelism



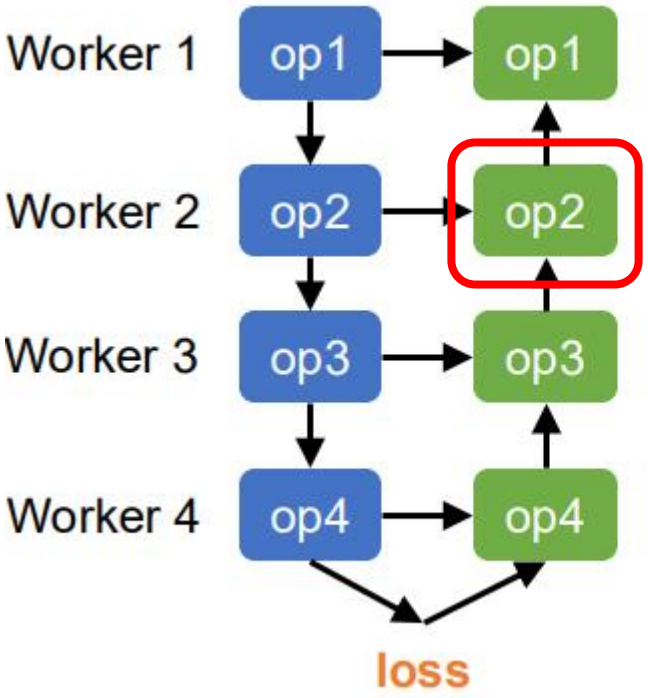


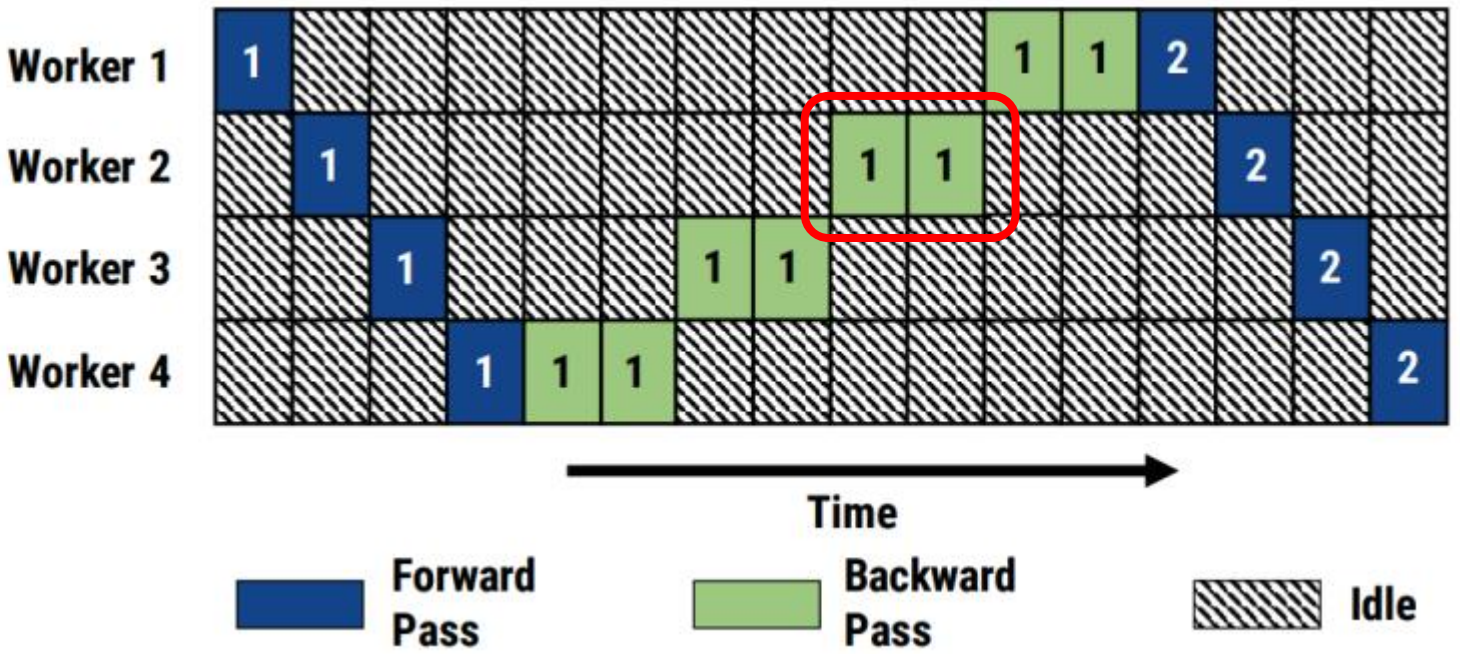
Simple Inter-Layer Parallelism



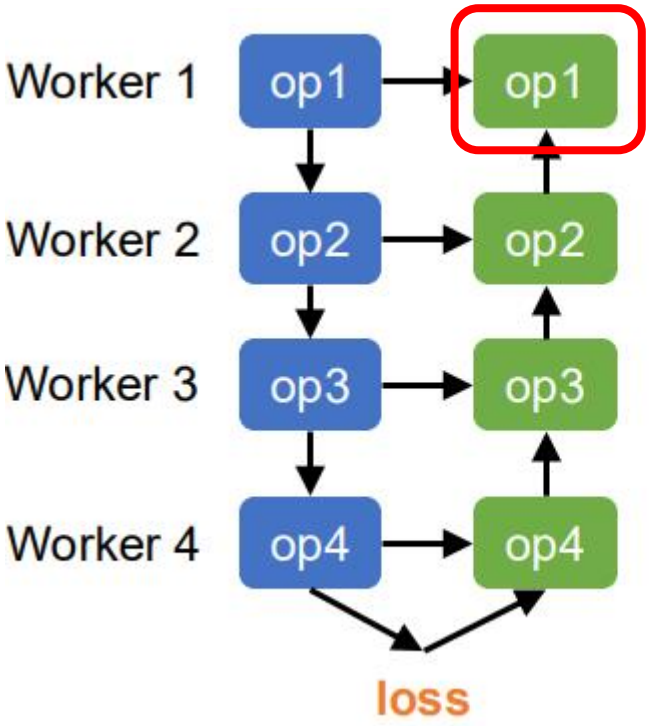


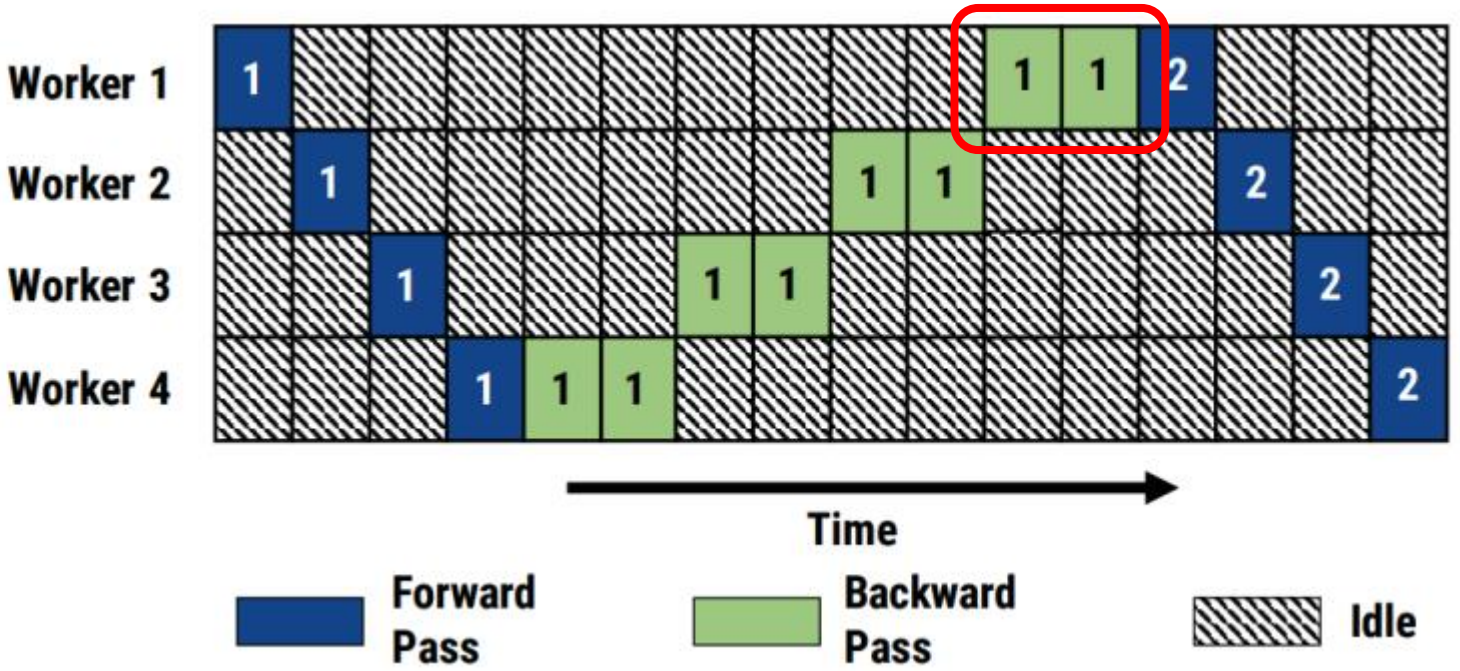
Simple Inter-Layer Parallelism



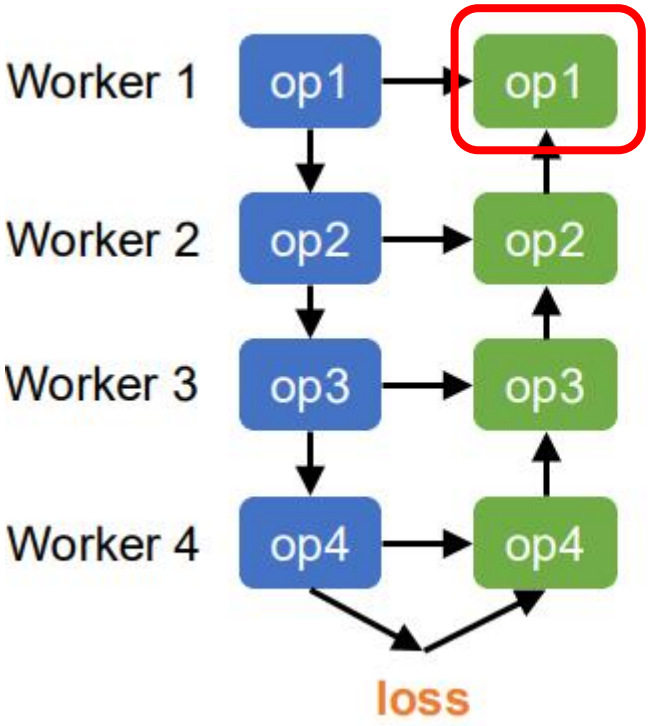


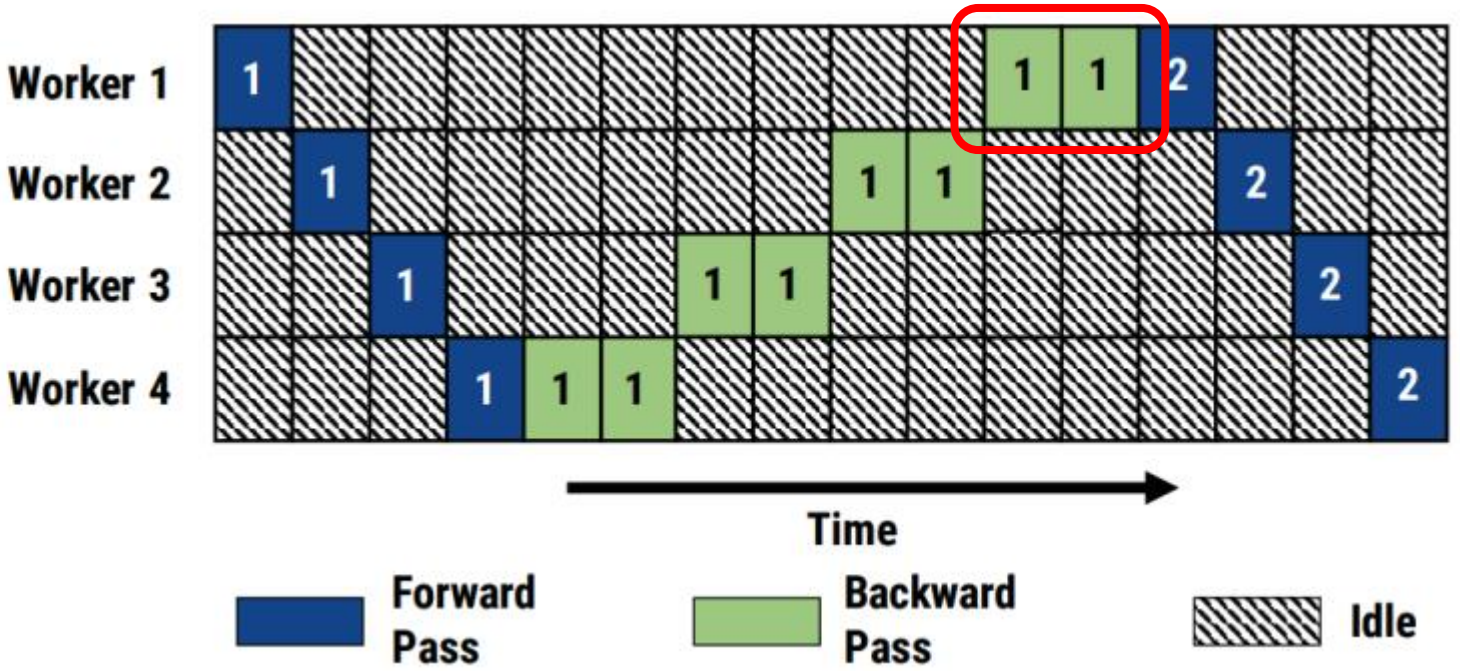
Simple Inter-Layer Parallelism





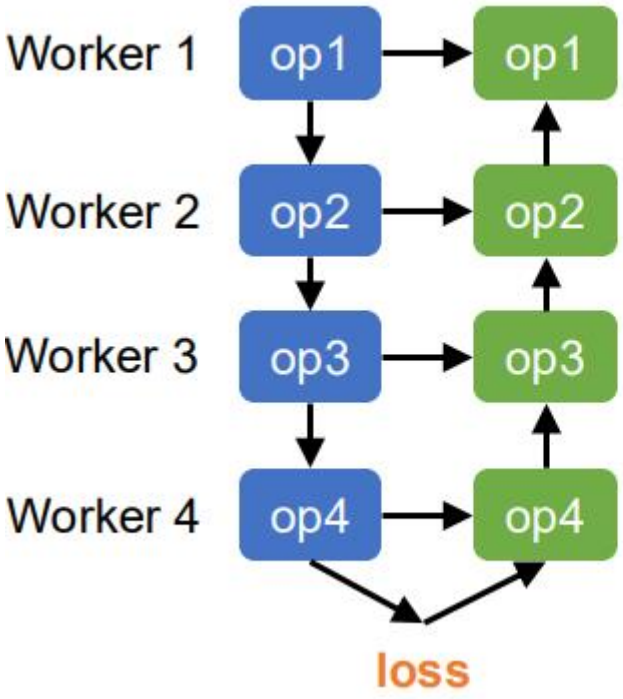
Simple Inter-Layer Parallelism

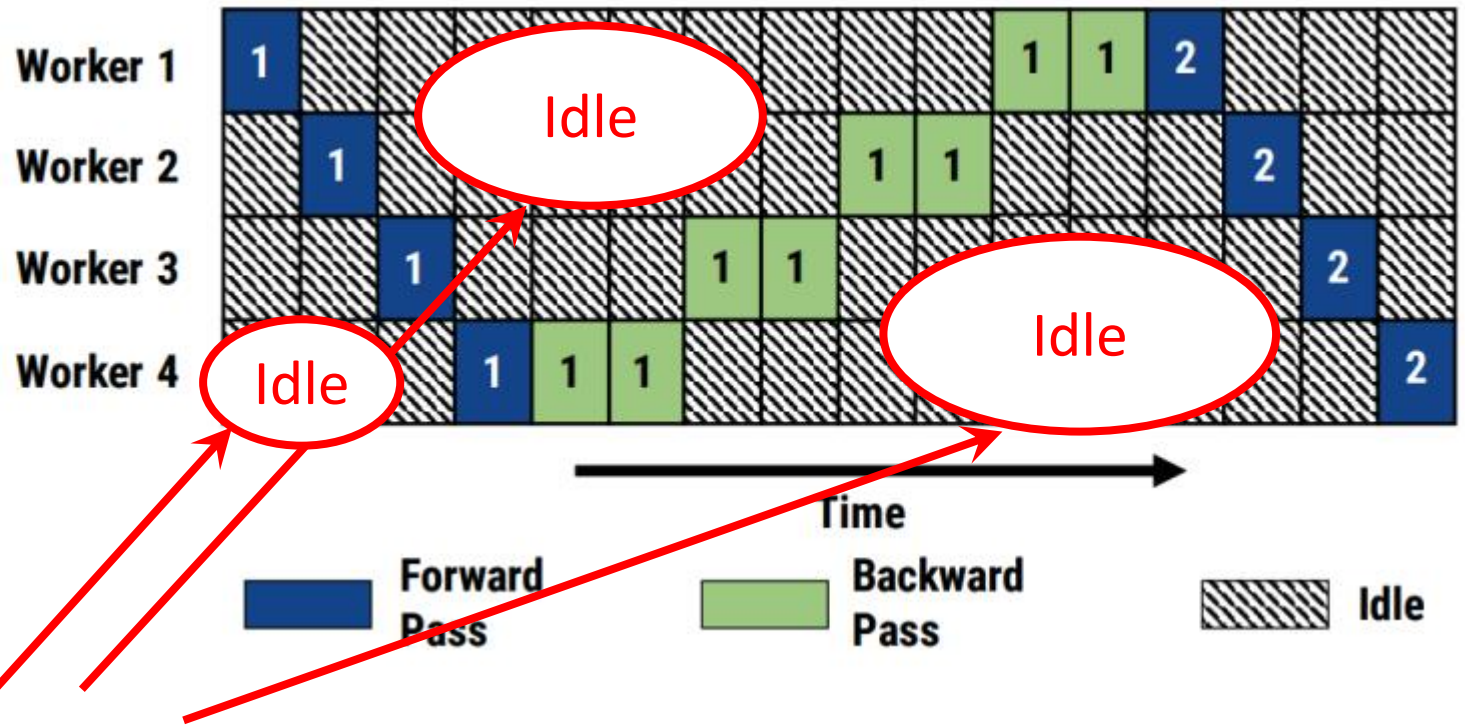




Question: What is the limitation of this execution?

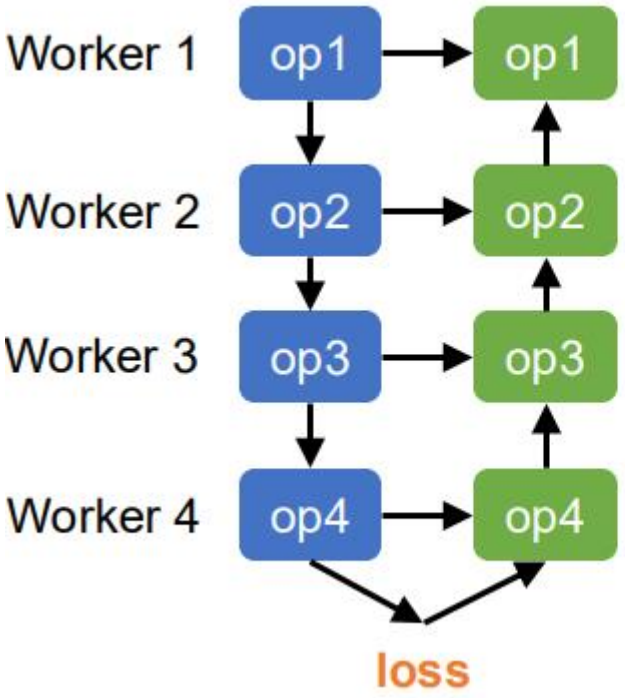
Issues with Simple Inter-Layer Parallelism

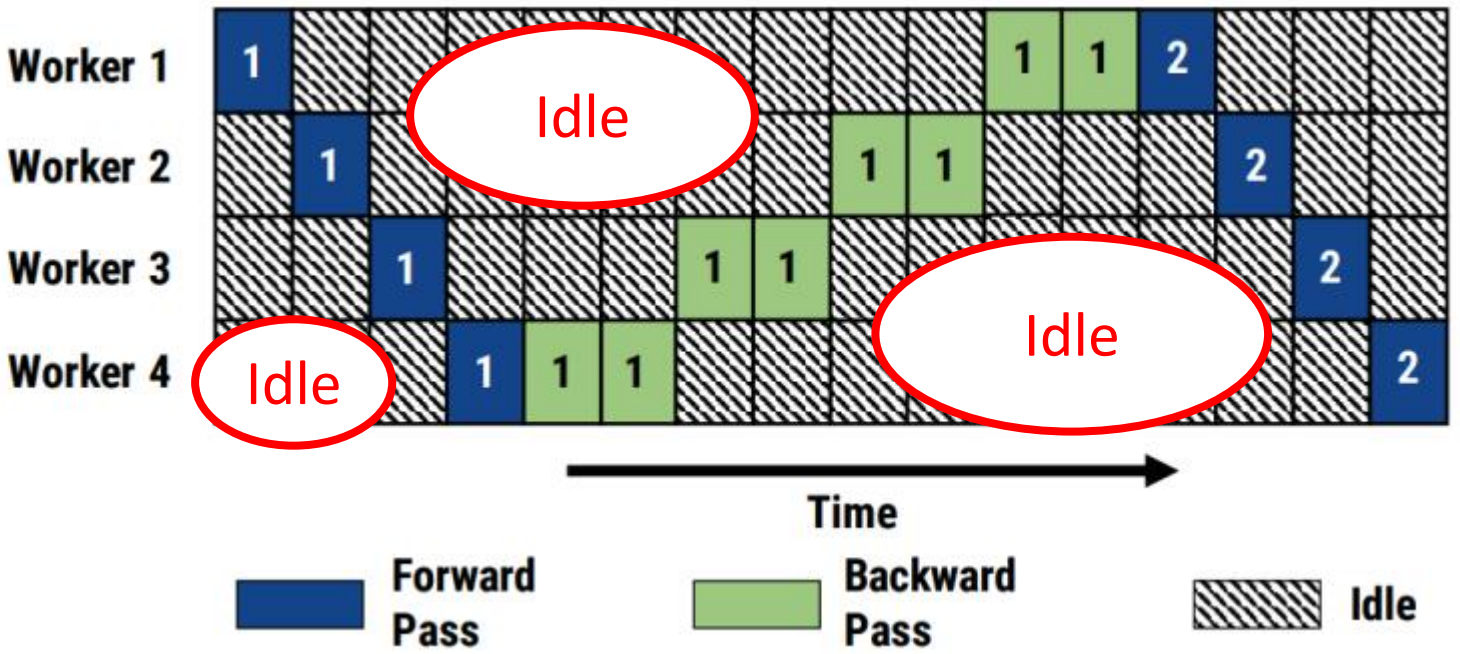




- Under-utilization of compute resources
- Only one device is computing at a time and others are idling

Issues with Simple Inter-Layer Parallelism



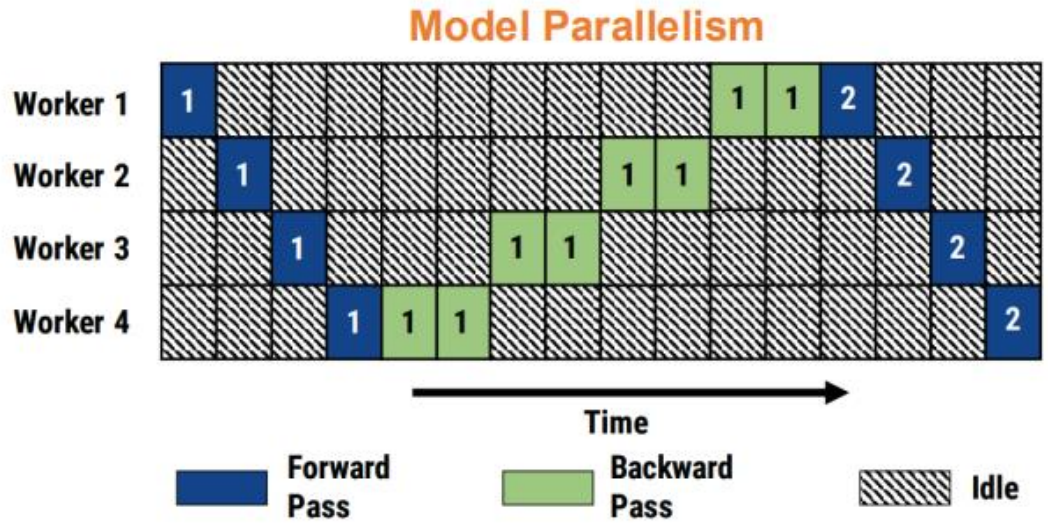


Question: How to improve the utilization and let multiple workers work simultaneously?

Pipeline Model Parallelism

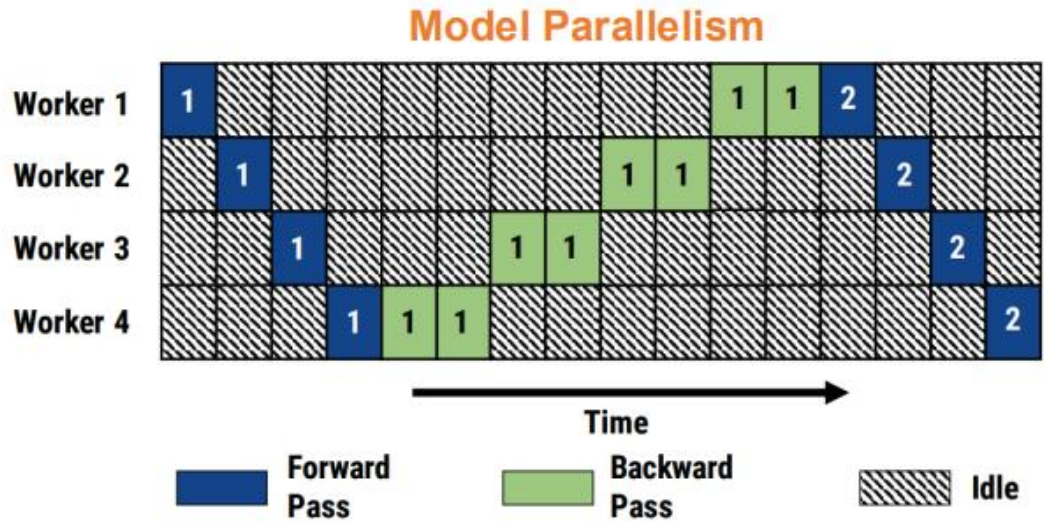
- Mini-batch: the number of samples processed in each iteration

- Divide a mini-batch into multiple smaller micro-batches



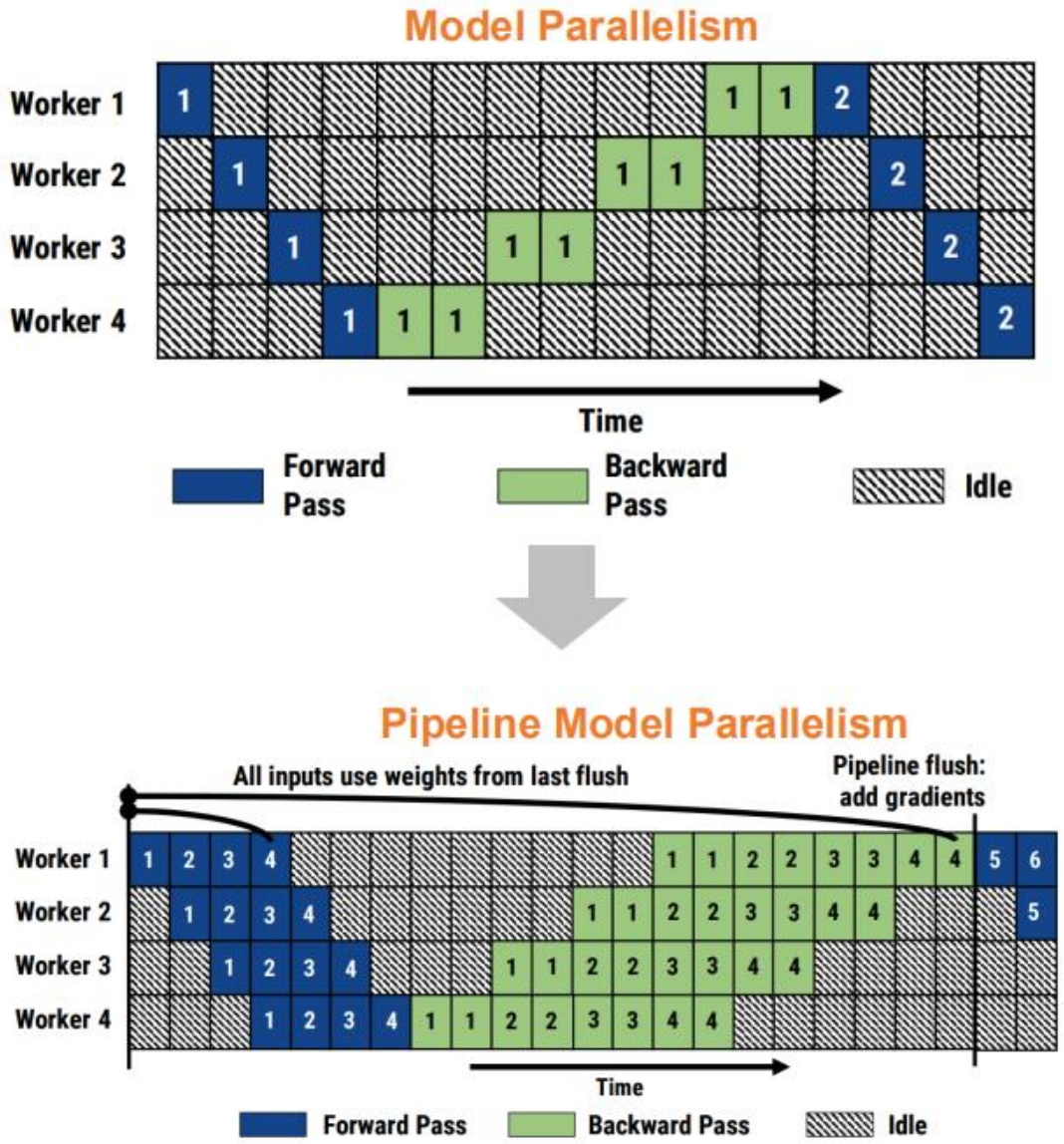
Pipeline Model Parallelism

- Mini-batch: the number of samples processed in each iteration
- Divide a mini-batch into multiple smaller micro-batches
- Pipeline execution:
  - Micro-batches flow through the pipeline from one stage to the next.
  - As soon as a stage completes its work, it passes the micro-batch to the next stage and starts working on the next micro-batch

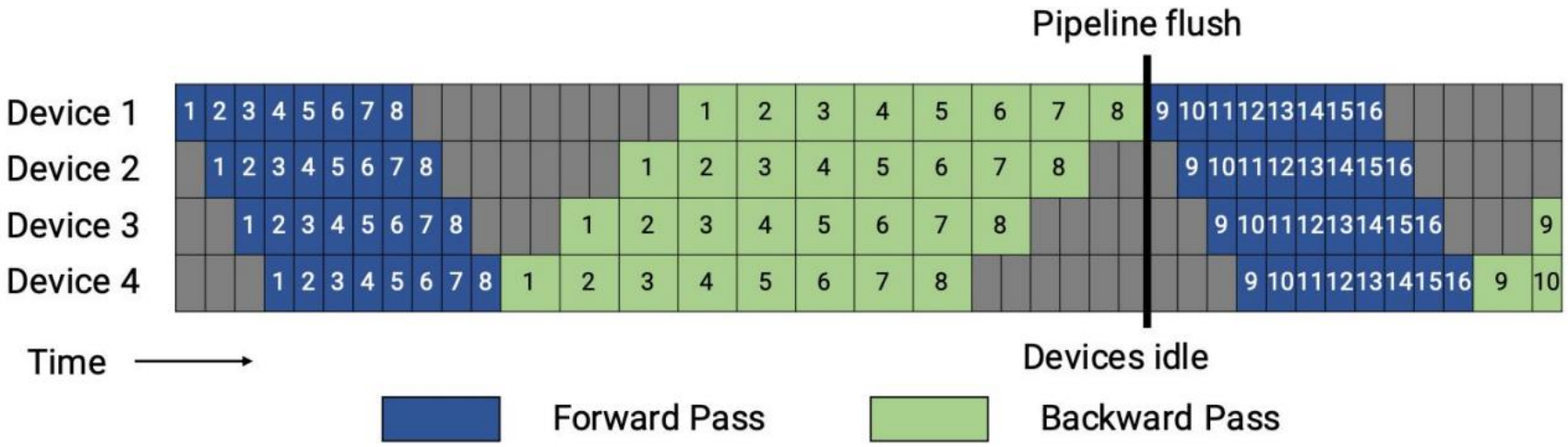


Pipeline Model Parallelism

- Mini-batch: the number of samples processed in each iteration
- Divide a mini-batch into multiple smaller micro-batches
- Pipeline execution:
  - Micro-batches flow through the pipeline from one stage to the next.
  - As soon as a stage completes its work, it passes the micro-batch to the next stage and starts working on the next micro-batch



Pipeline Model Parallelism: Device Utilization

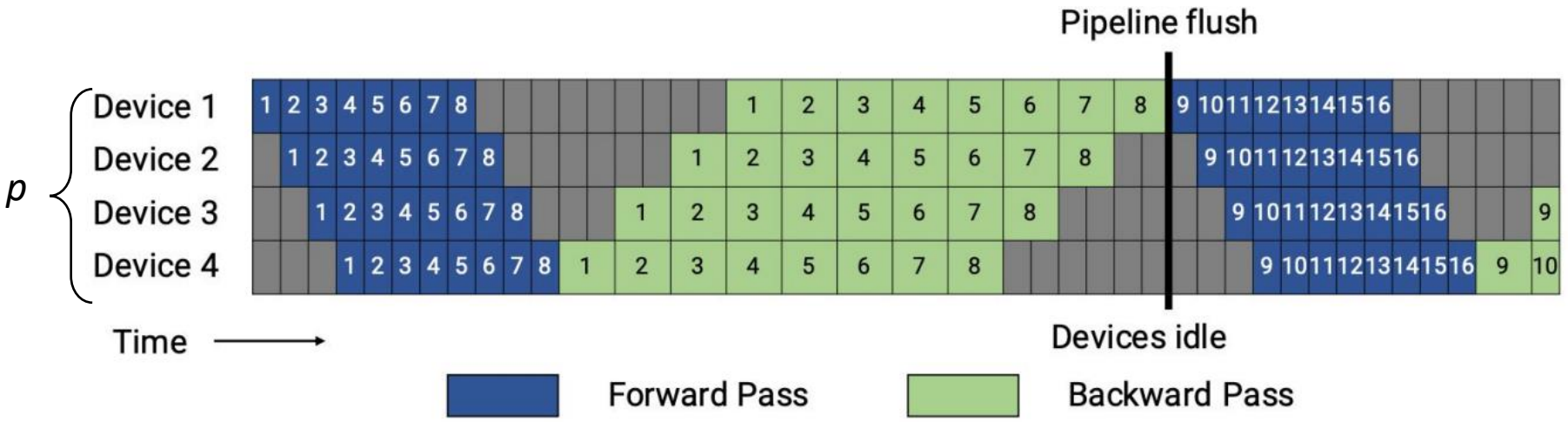


Still have bubbles in the pipeline

Question: How do we quantify bubbles?

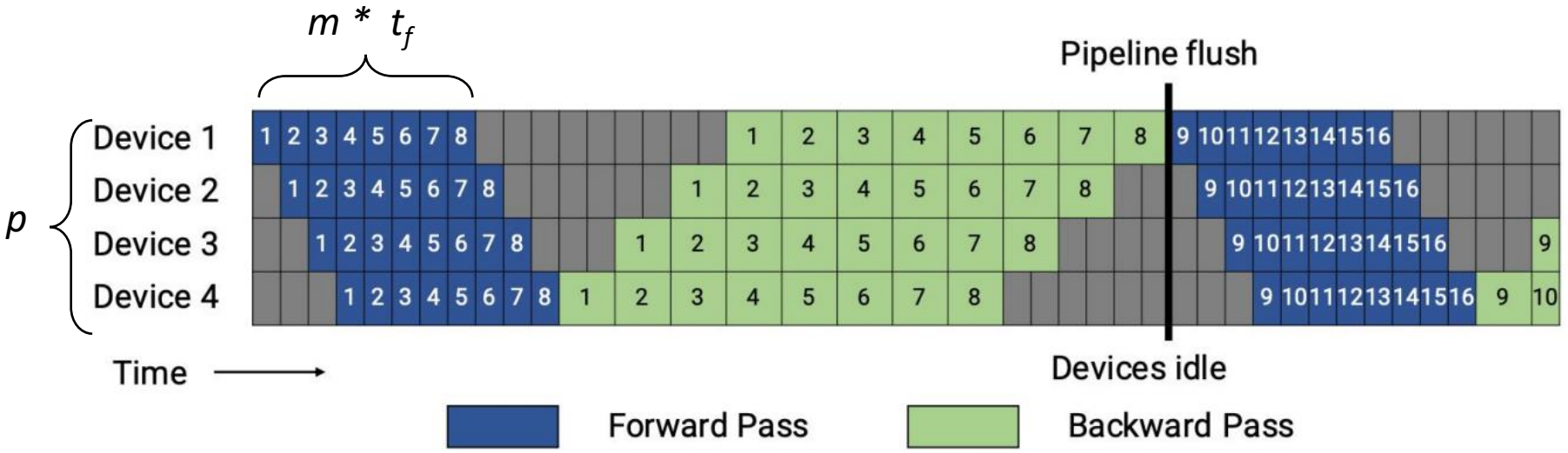
Pipeline Model Parallelism: Device Utilization

m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



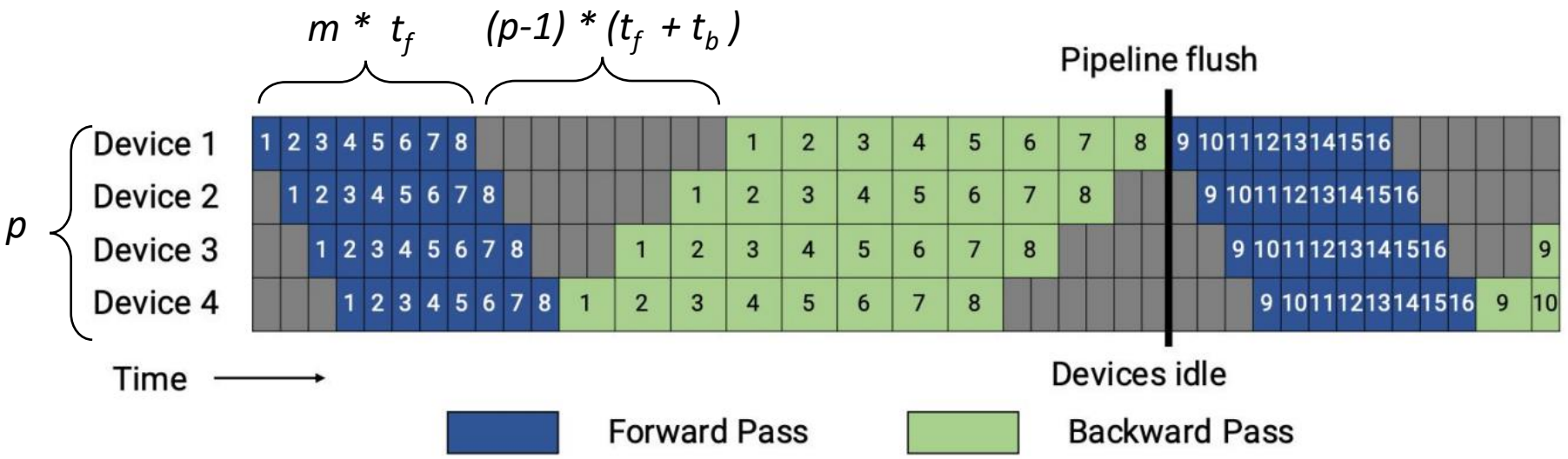
Pipeline Model Parallelism: Device Utilization

m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



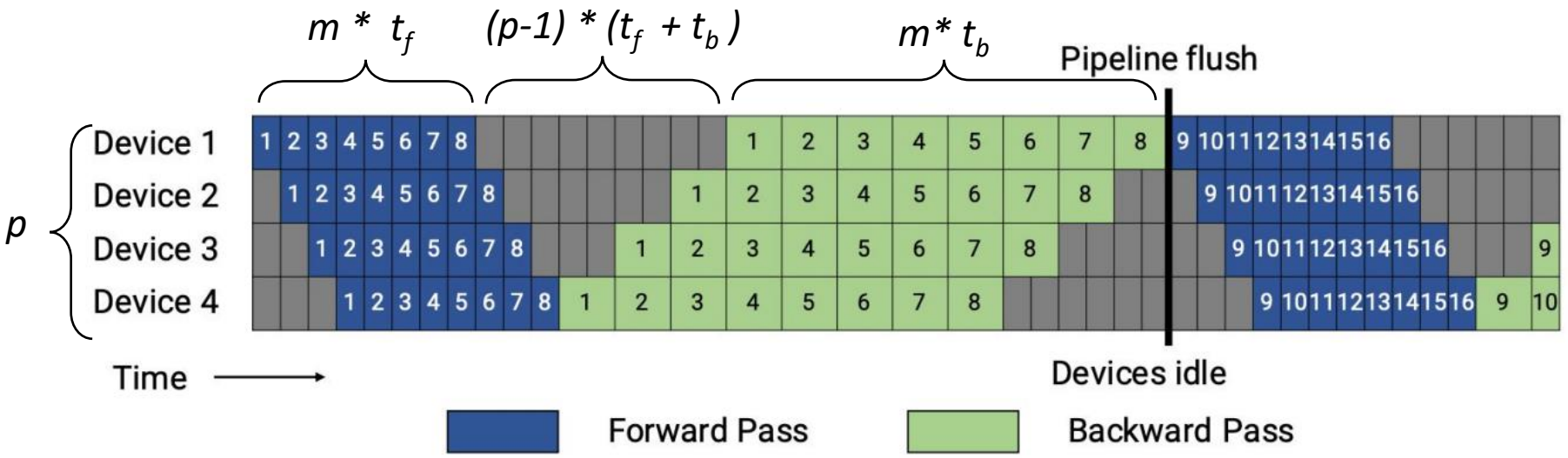
Pipeline Model Parallelism: Device Utilization

m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



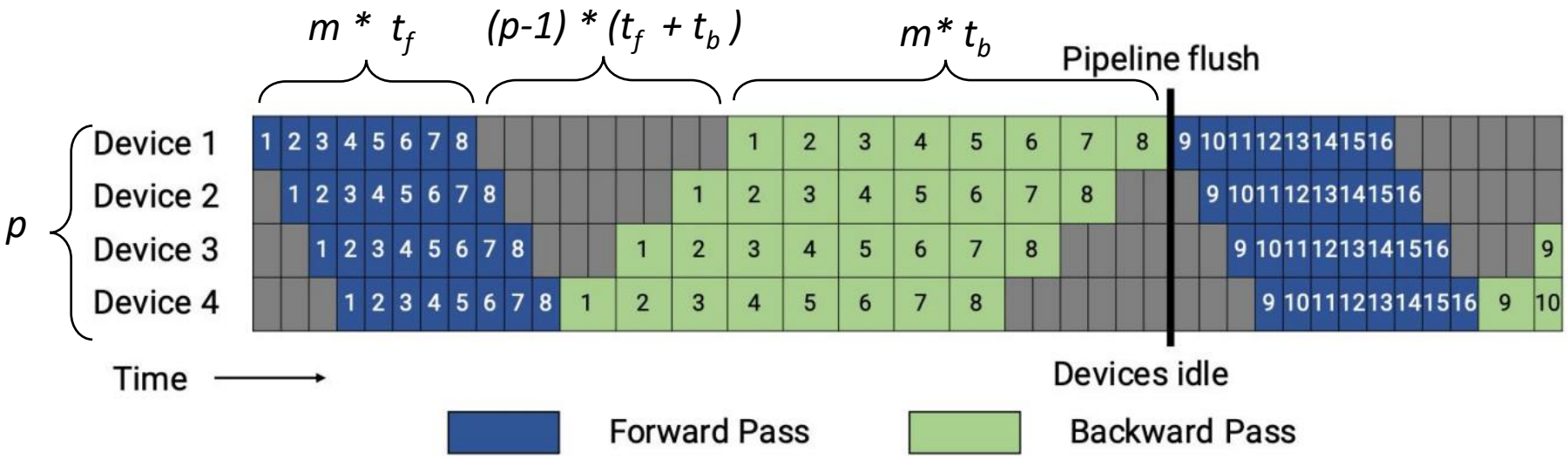
Pipeline Model Parallelism: Device Utilization

m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



Pipeline Model Parallelism: Device Utilization

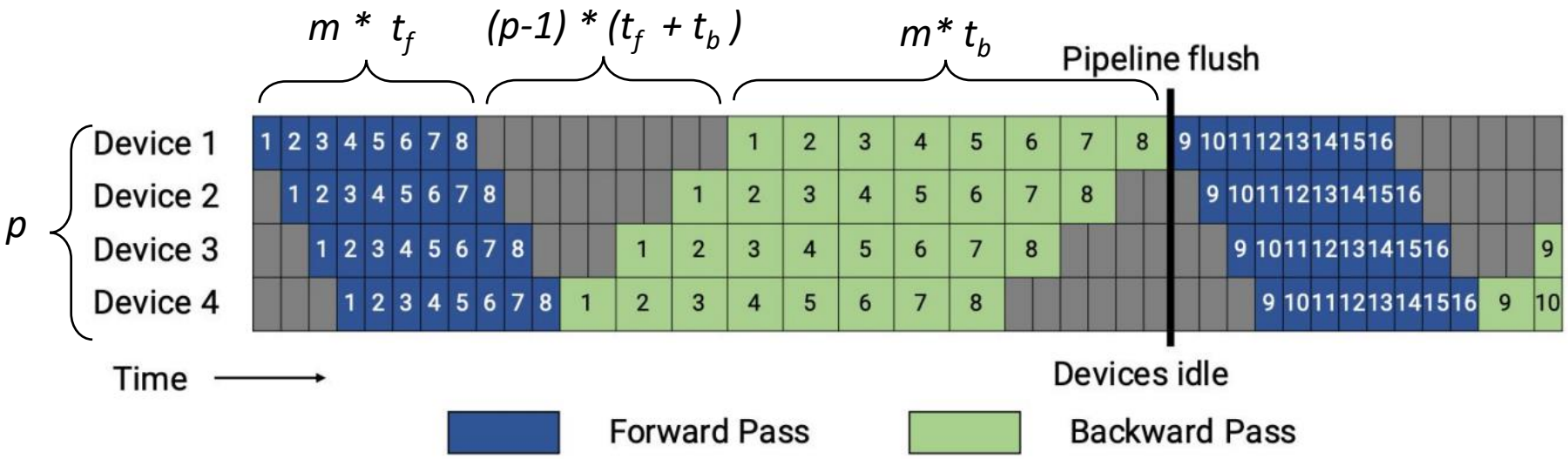
m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



$$\textbf{BubbleFraction} = \frac{(p-1) * (t_f + t_b)}{m * t_f + m * t_b} = \frac{p-1}{m}$$

Pipeline Model Parallelism: Device Utilization

m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward

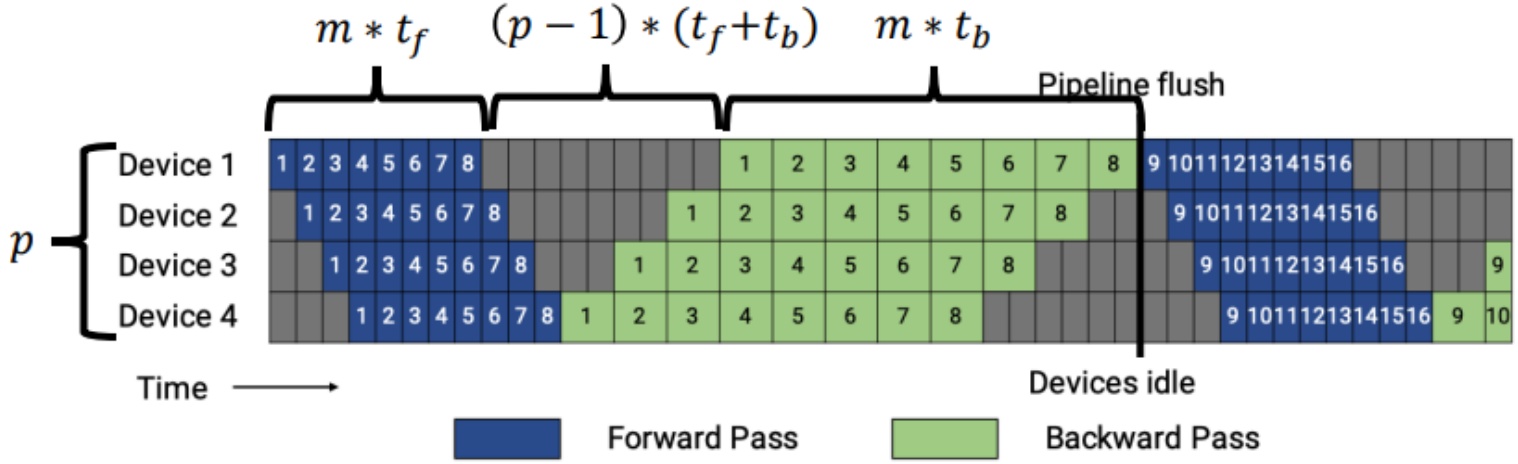


$$\textbf{BubbleFraction} = \frac{(p-1) * (t_f + t_b)}{m * t_f + m * t_b} = \frac{p-1}{m}$$

Question: How do we reduce the bubble fraction?

Improving Pipeline Parallelism Efficiency

- m : number of micro-batches in a mini-batch
  - Increase mini-batch size or reduce micro-batch size
  - Caveat: large mini-batch sizes can lead to accuracy loss; small micro-batch sizes reduce GPU utilization
- p: number of pipeline stages
  - Decrease pipeline depth
  - Caveat: increase stage size



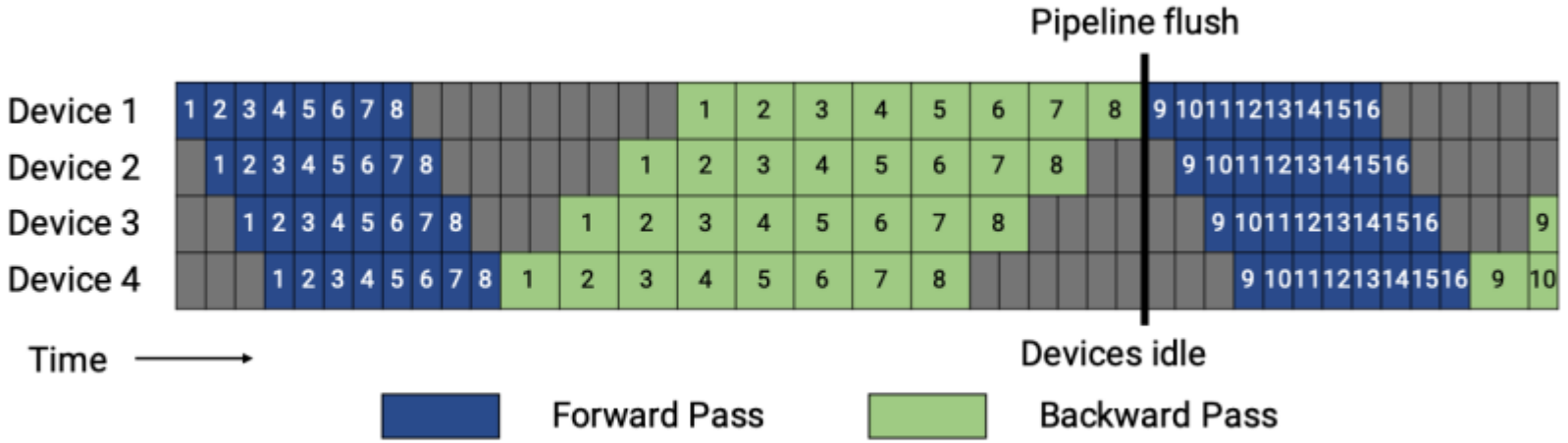
$$\textbf{BubbleFraction} = \frac{(p-1) * (t_f+t_b)}{m * t_f + m * t_b} = \frac{p-1}{m}$$

Design More Advanced Pipeline Schedule

- Each machine makes a choice between two options:
  - Perform the forward pass for a micro-batch, pushing the micro-batch to downstream workers
  - Perform the backward pass for a different micro-batch, ensuring forward progress in learning

Improving Pipeline Parallelism Efficiency

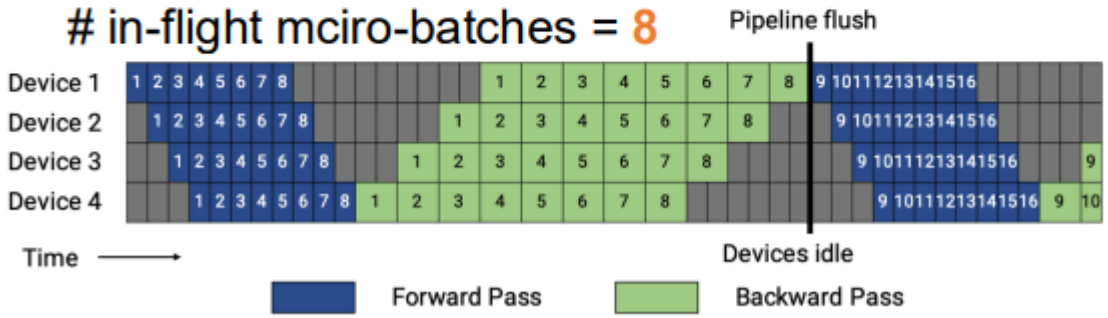
- An issue: we need to keep the intermediate activations of **all micro-batches** before back propagation



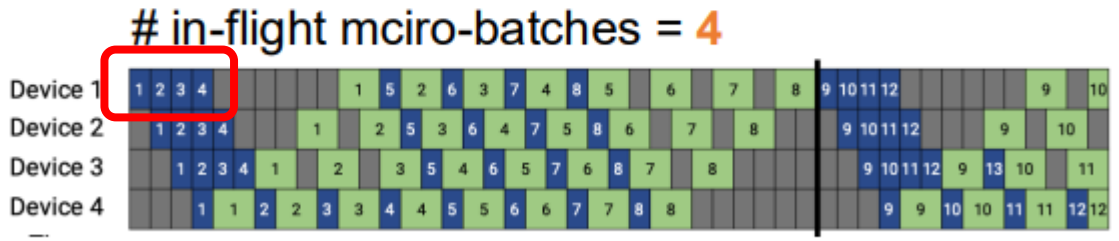
Question: Can we improve the pipeline schedule to reduce memory requirements?

Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state



Pipeline parallelism with GPipe's schedule

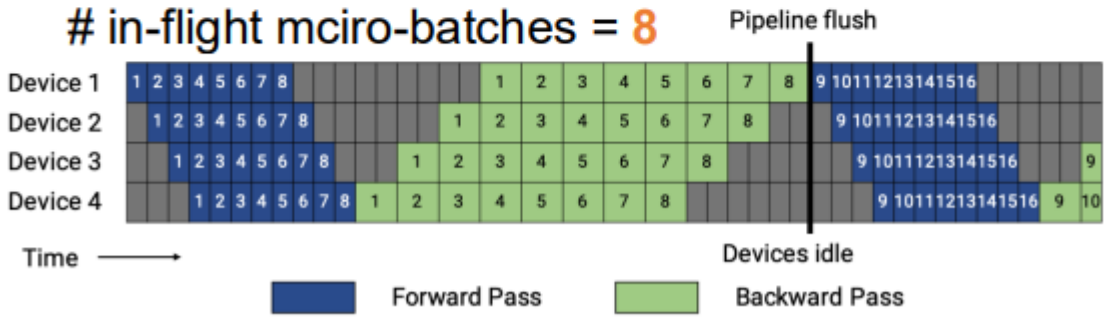


Pipeline parallelism with 1F1B schedule

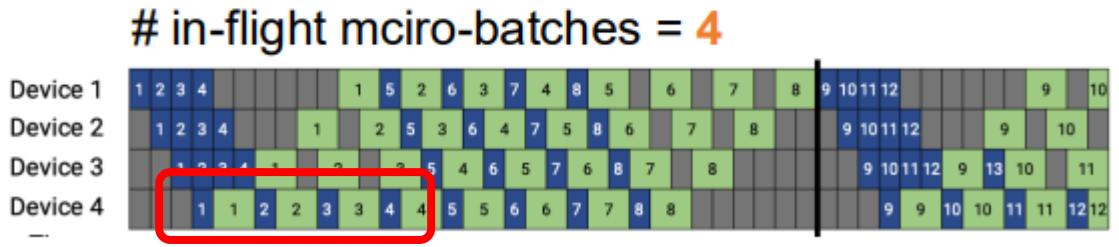
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state



Pipeline parallelism with GPipe's schedule

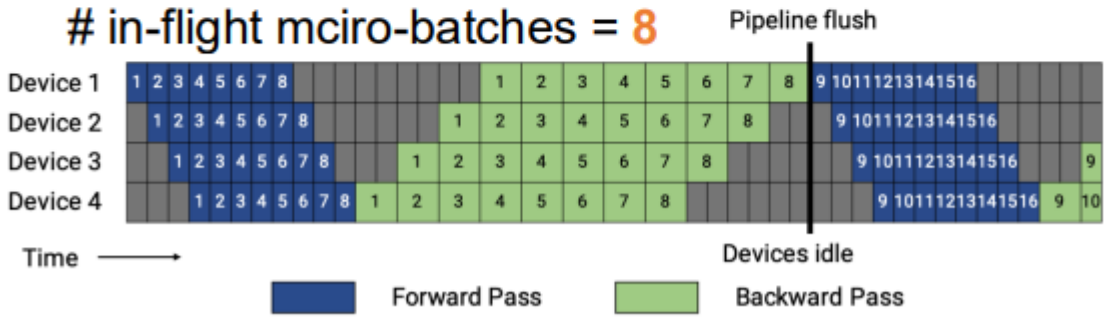


Pipeline parallelism with 1F1B schedule

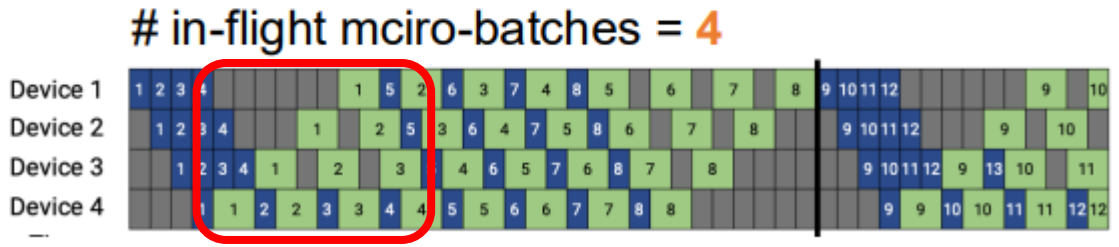
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state



Pipeline parallelism with GPipe's schedule

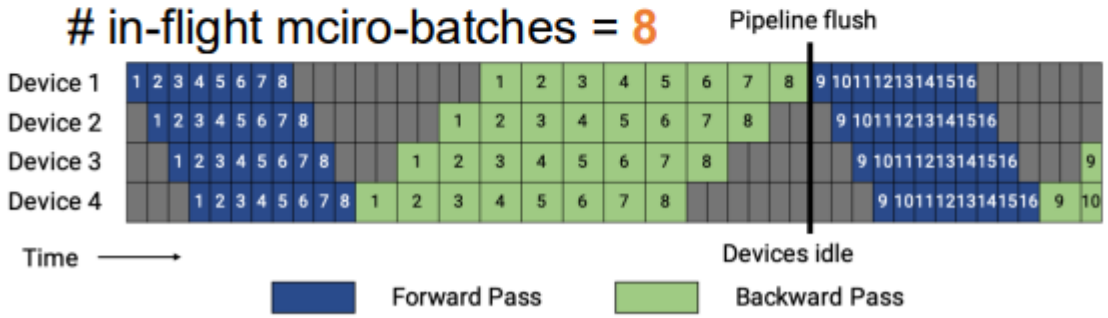


Pipeline parallelism with 1F1B schedule

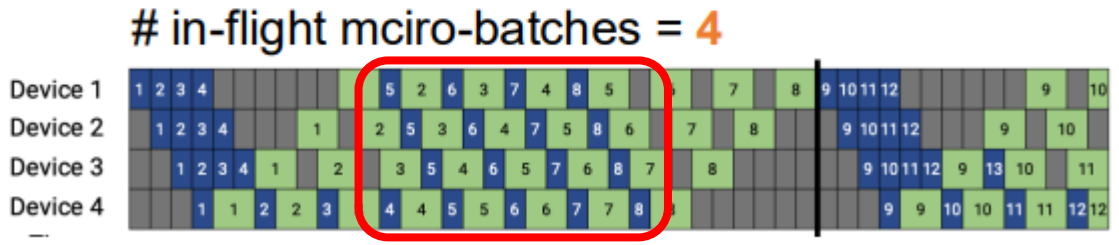
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state



Pipeline parallelism with GPipe's schedule

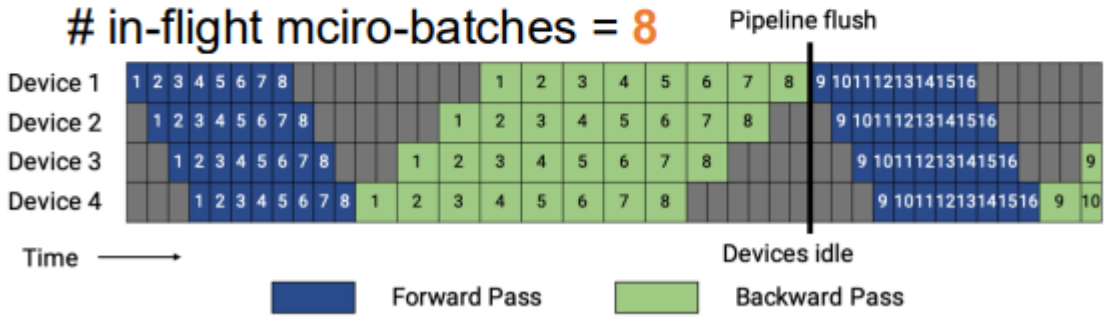


Pipeline parallelism with 1F1B schedule

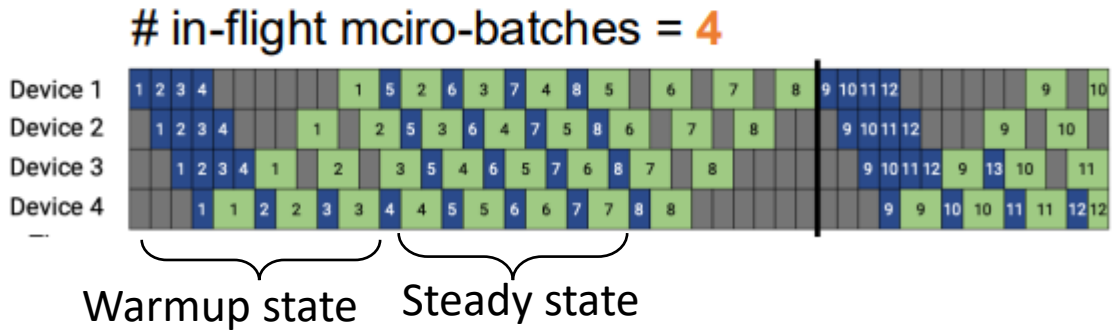
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state



Pipeline parallelism with GPipe's schedule



Pipeline parallelism with 1F1B schedule

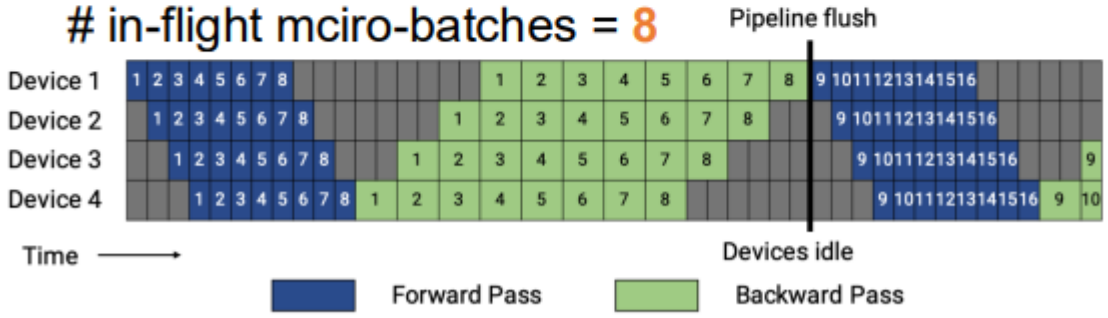
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

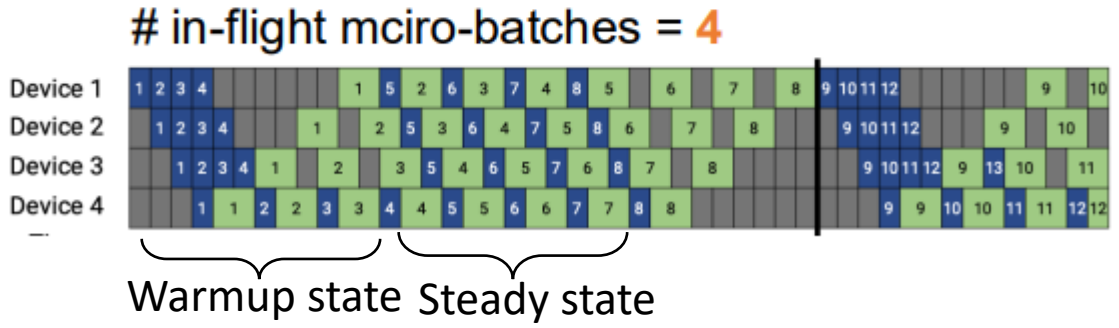
- One-Forward-One-Backward in the steady state

- **Reduce memory footprint of pipeline parallelism**
- **Doesn't reduce pipeline bubble**

Can we reduce pipeline bubble?



Pipeline parallelism with GPipe's schedule

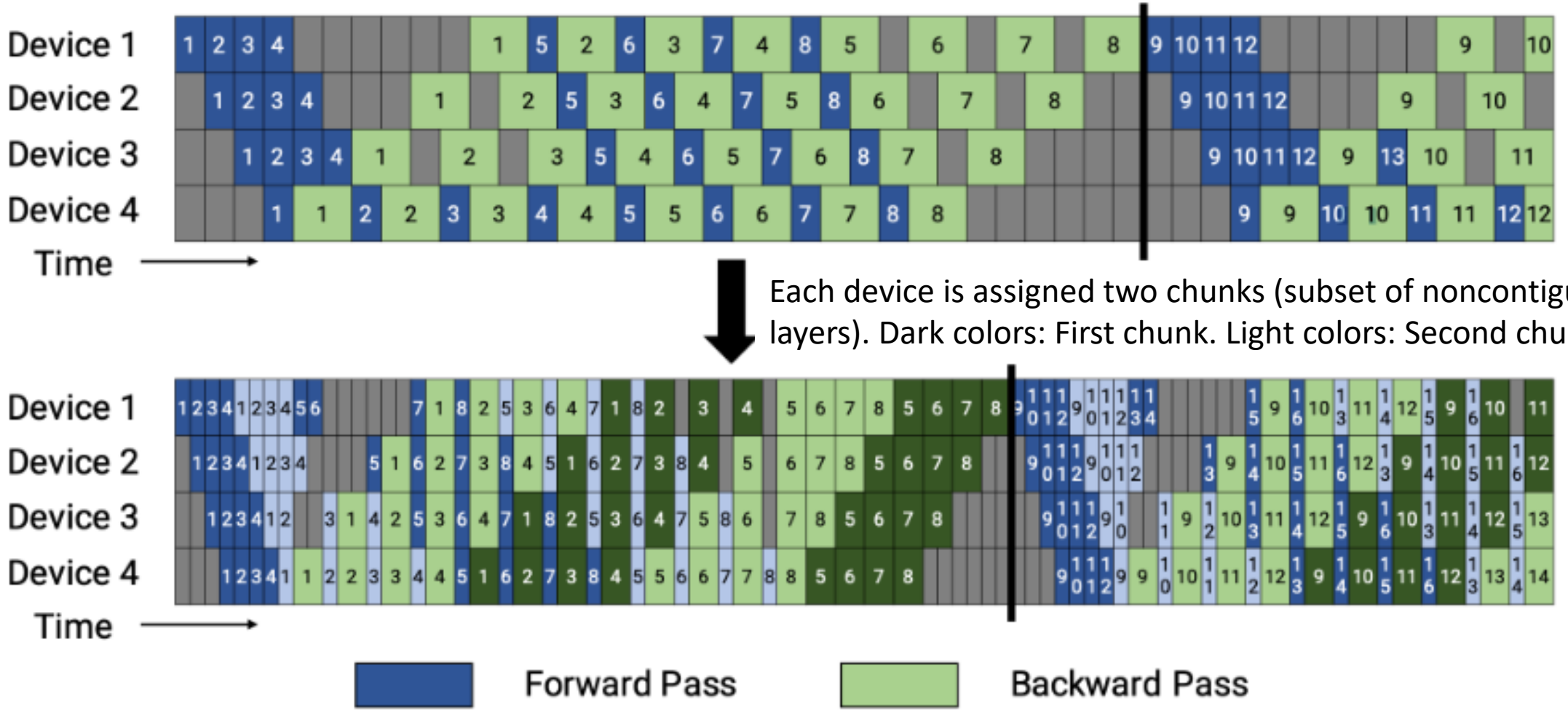


Pipeline parallelism with 1F1B schedule

PipeDream: Fast and Efficient Pipeline Parallel DNN Training

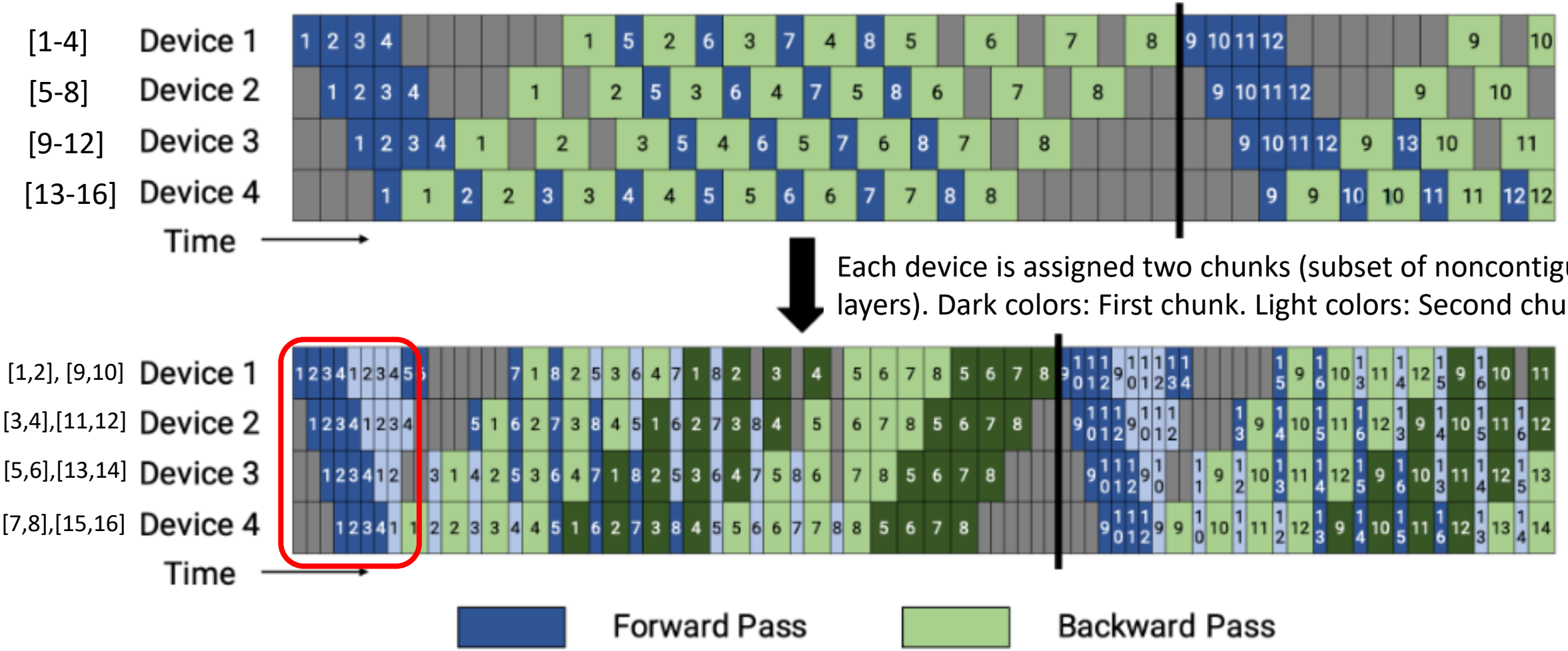
Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is t/v



Pipeline Parallelism with Interleaved 1F1B Schedule

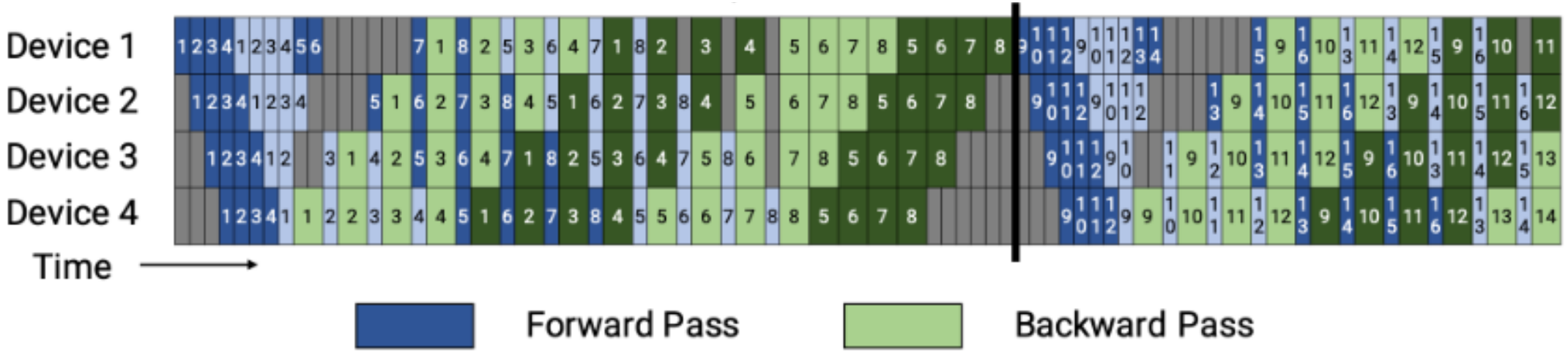
- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is t/v



Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is t/v

$$BubbleFraction = \frac{(p-1) * \frac{(t_f + t_b)}{v}}{m * t_f + m * t_b} = \frac{1}{v} * \frac{p-1}{m}$$

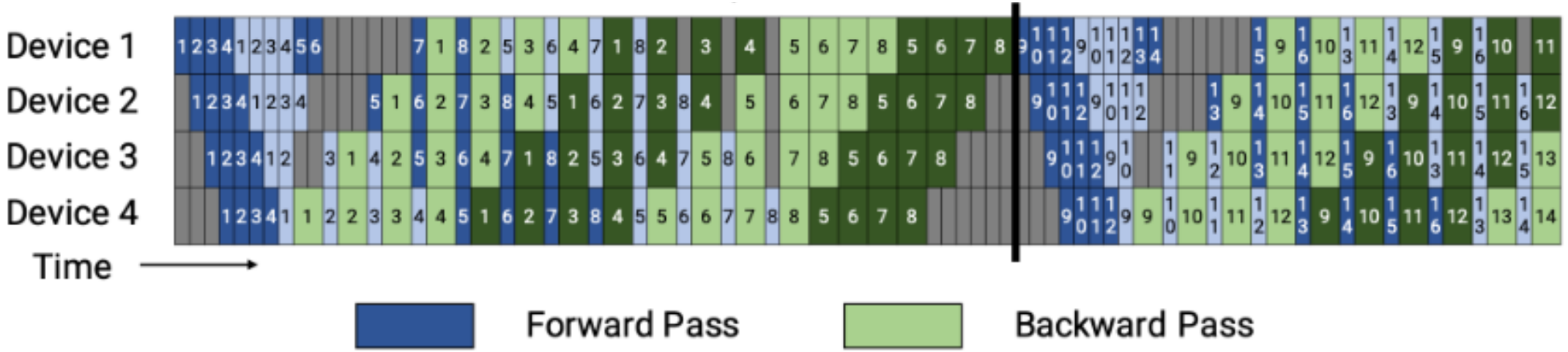


Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is t/v

$$BubbleFraction = \frac{(p-1) * \frac{(t_f + t_b)}{v}}{m * t_f + m * t_b} = \frac{1}{v} * \frac{p-1}{m}$$

Question: Increasing v improves pipeline efficiency?

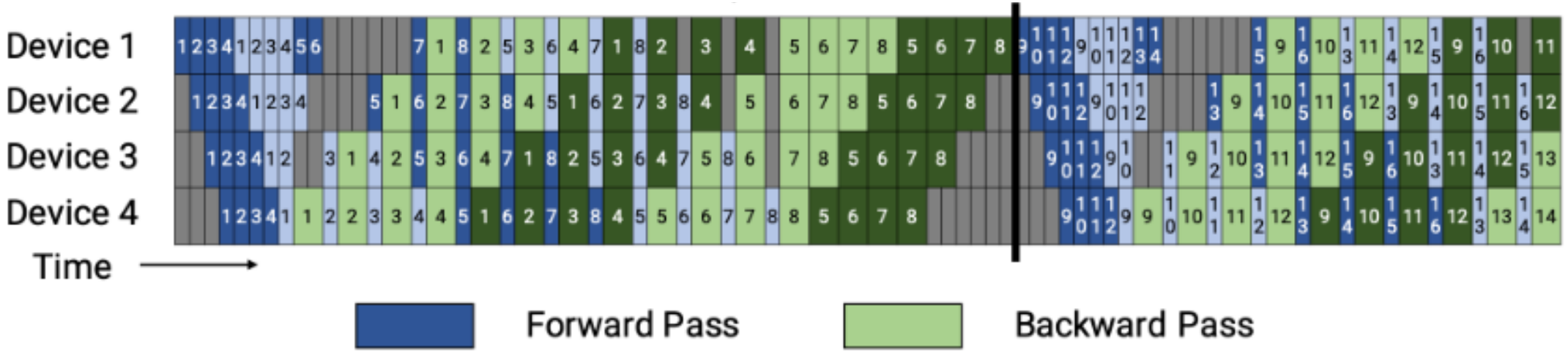


Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is t/v

$$BubbleFraction = \frac{(p-1) * \frac{(t_f + t_b)}{v}}{m * t_f + m * t_b} = \frac{1}{v} * \frac{p-1}{m}$$

Reduce bubble time at the cost of increased communication



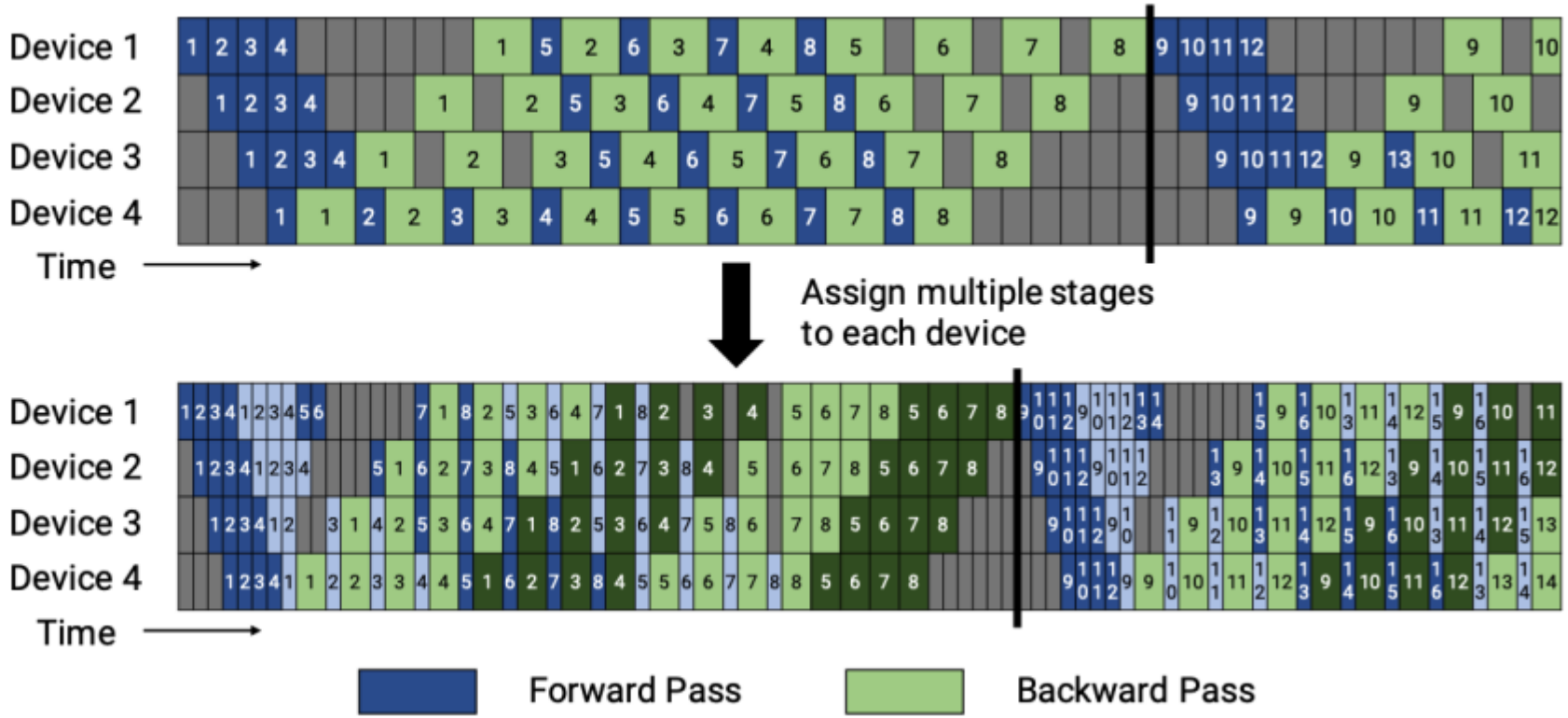
Pipeline Parallelism with Interleaved 1F1B Schedule

Pipeline parallelism with 1F1B Schedule

$$BubbleFraction = \frac{p-1}{m}$$

Pipeline parallelism with interleaved 1F1B Schedule

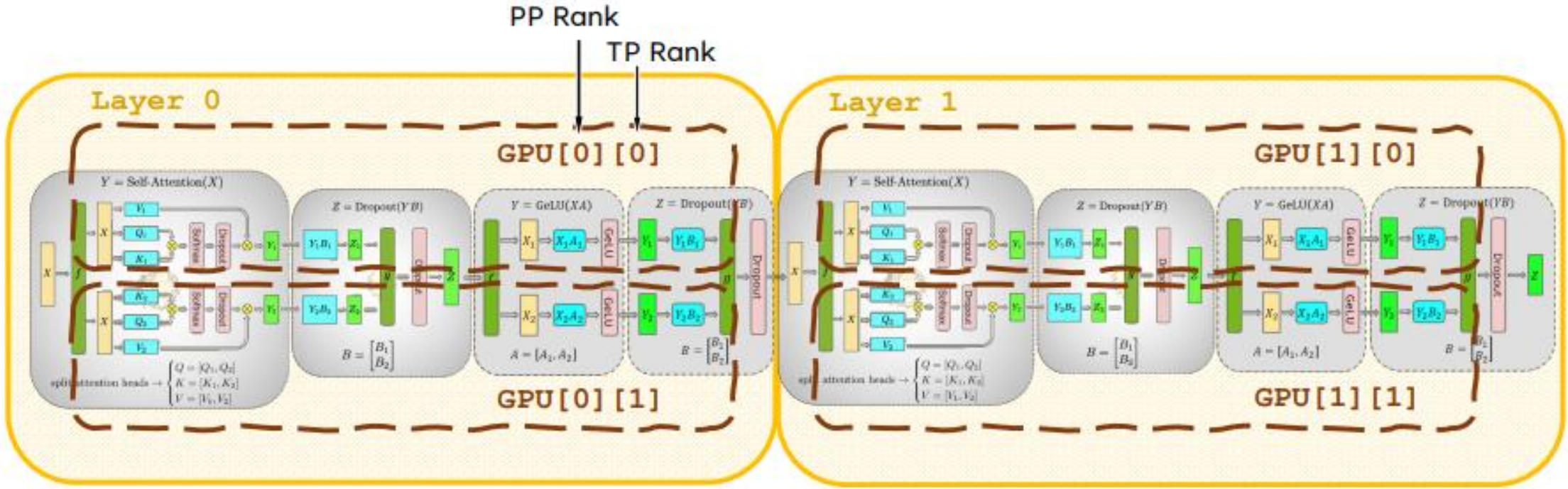
$$BubbleFraction = \frac{1}{v} * \frac{p-1}{m}$$



Combining Multiple Parallelism

Combining Multiple Parallelism

- Model_Parallelism = Tensor_Parallelism × Pipeline_Parallelism



Performance Analysis of Combined Parallelism

- Tensor and Pipeline Model Parallelism
  - t ⇧, pipeline bubble ⇩

$$\frac{p-1}{m} = \frac{n/t-1}{m}$$

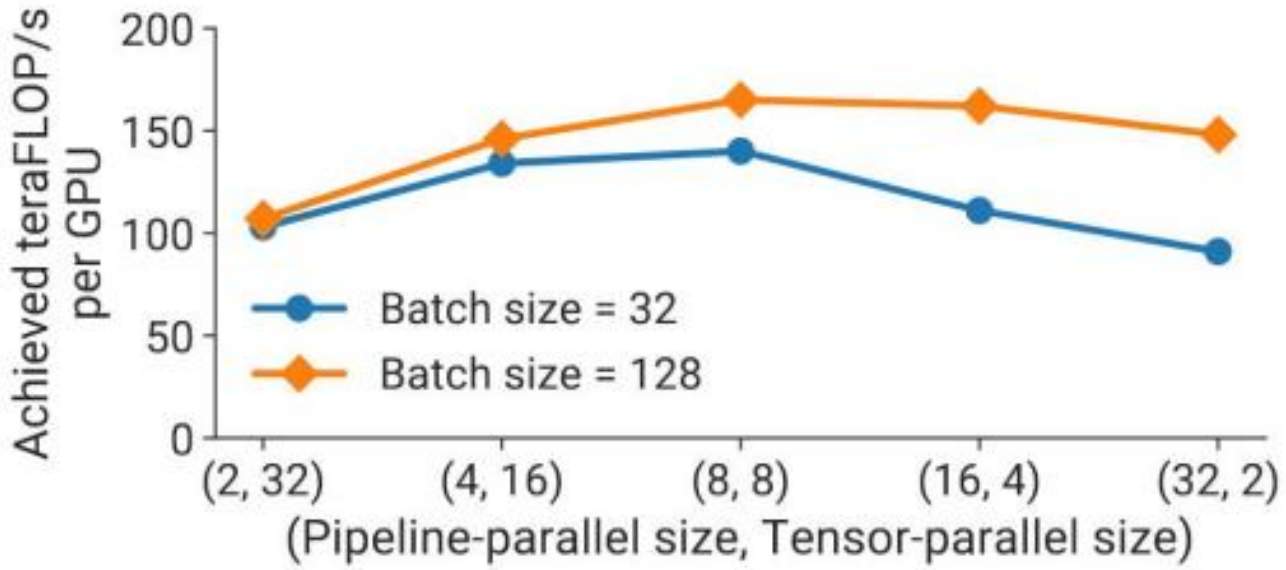
- (p, t, d): Parallelization dimensions, where p is the pipeline-model-parallel size, t is the tensor-model-parallel size, and d is the data-parallel size.
- n: Number of GPUs, satisfying $p \cdot t \cdot d = n$.
- B: Global batch size.
- b: Microbatch size.
- $m = \frac{B}{b \cdot d}$: Number of microbatches per pipeline.

- Communication overhead
  - All-reduce communication for tensor model parallelism is expensive!
  - Especially when cross servers

Takeaway #1: Use tensor model parallelism within a server and pipeline model parallelism to scale to multiple servers.

Evaluation – TP vs. PP

- Tensor versus Pipeline Parallelism
  - 161-billion param. GPT
  - Peak performance achieved when t = p = 8
  - Need a conjunction of both types of model parallelisms

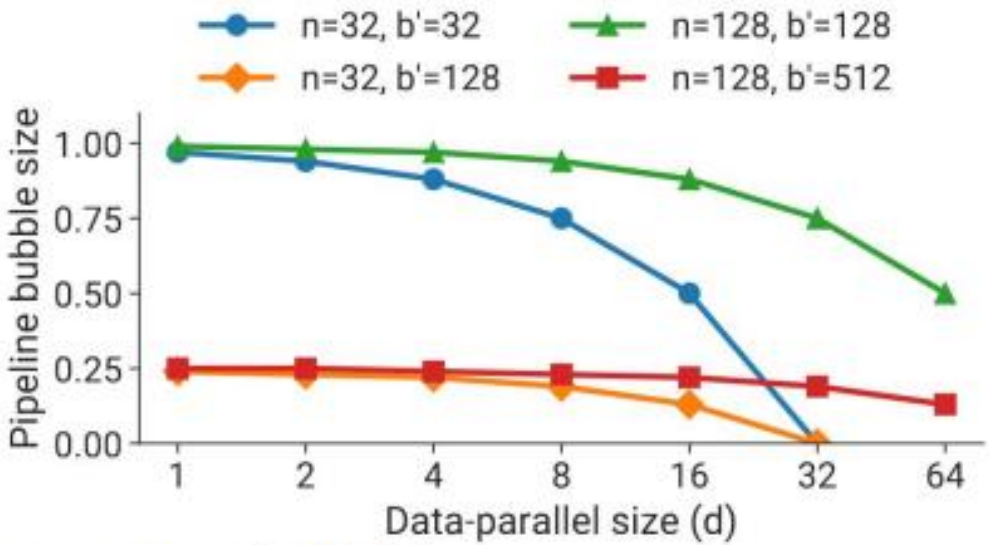


Performance Analysis of Combined Parallelism

- Data versus Pipeline Parallelism

$$\frac{p-1}{m} = \frac{n/d-1}{b'/d} = \frac{n-d}{b'} \quad b'=B/b$$

*m = B / (d * b) = b' / d*

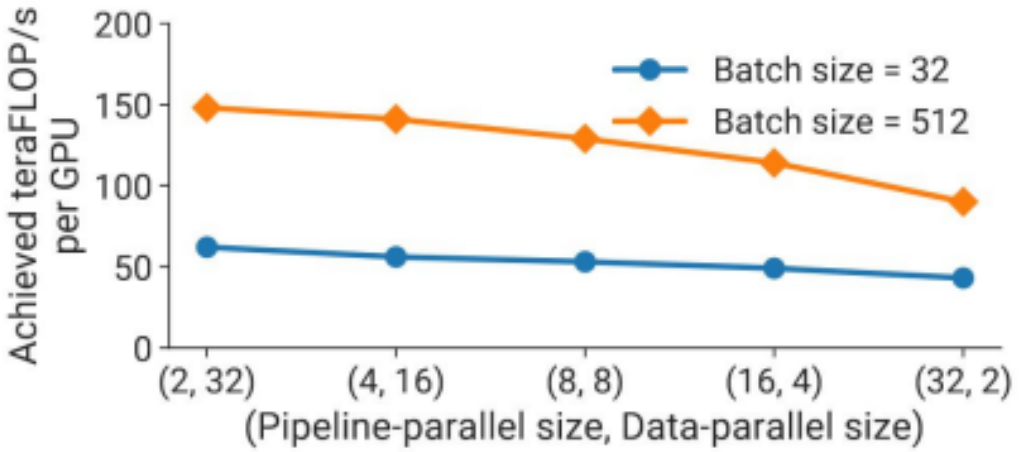


- Data versus Tensor Parallelism
  - DP is less communication heavy than TP
    - All-reduce once per batch vs. All-reduce once per microbatch
  - Tensor parallelism can lead to hardware underutilization

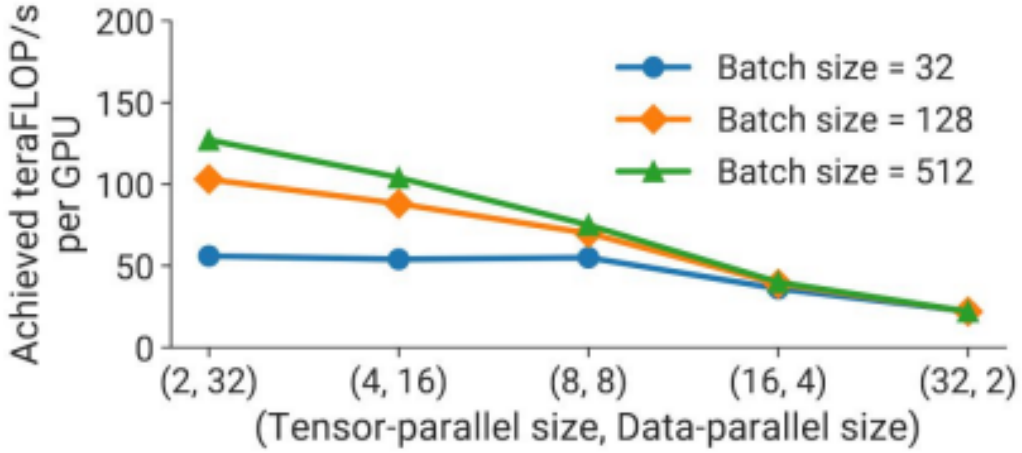
Takeaway #2: Decide tensor-parallel size and pipeline-parallel size based on the GPU memory size; data parallelism can be used to scale to more GPUs.

Evaluation - DP vs. Model Parallelism

- Pipeline-parallelism vs. Data-parallelism
  - 5.9-billion param. GPT
  - Throughput decreases as pipeline-parallel size increases



- Tensor-parallelism vs. Data-parallelism
  - 5.9-billion param. GPT
  - Throughput decreases as tensor-parallel size increases

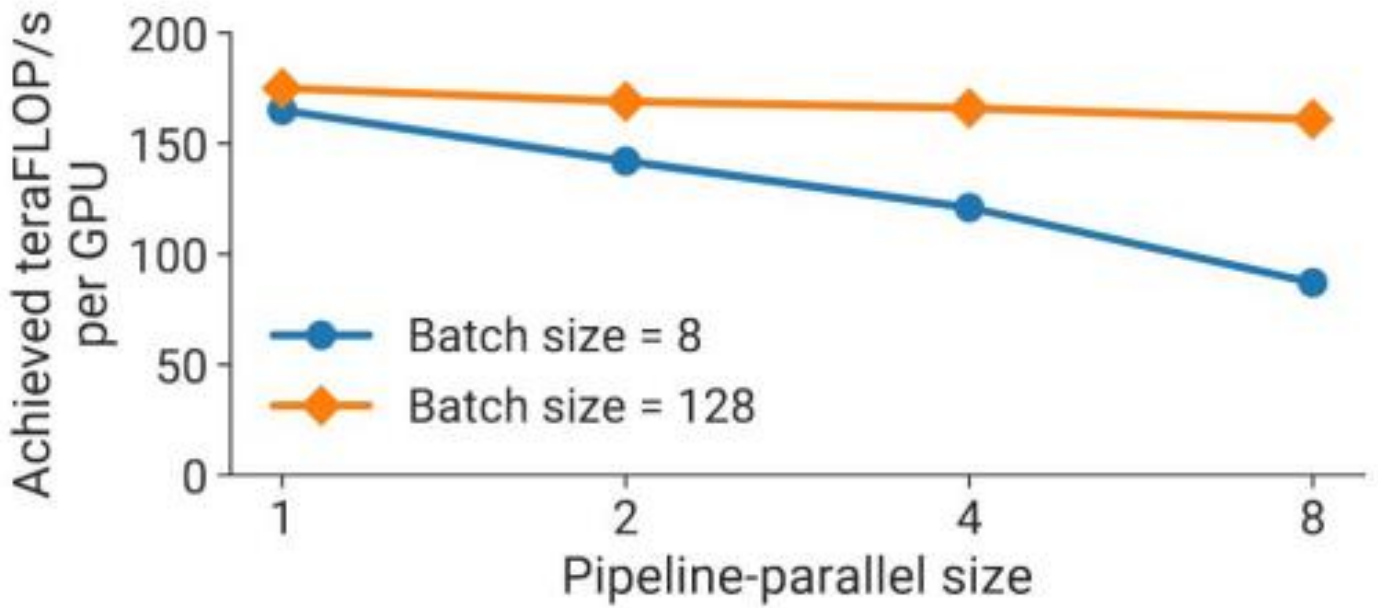


Limitations of data-parallelism:

1. Memory capacity
2. Scaling limitation proportional to the batch size

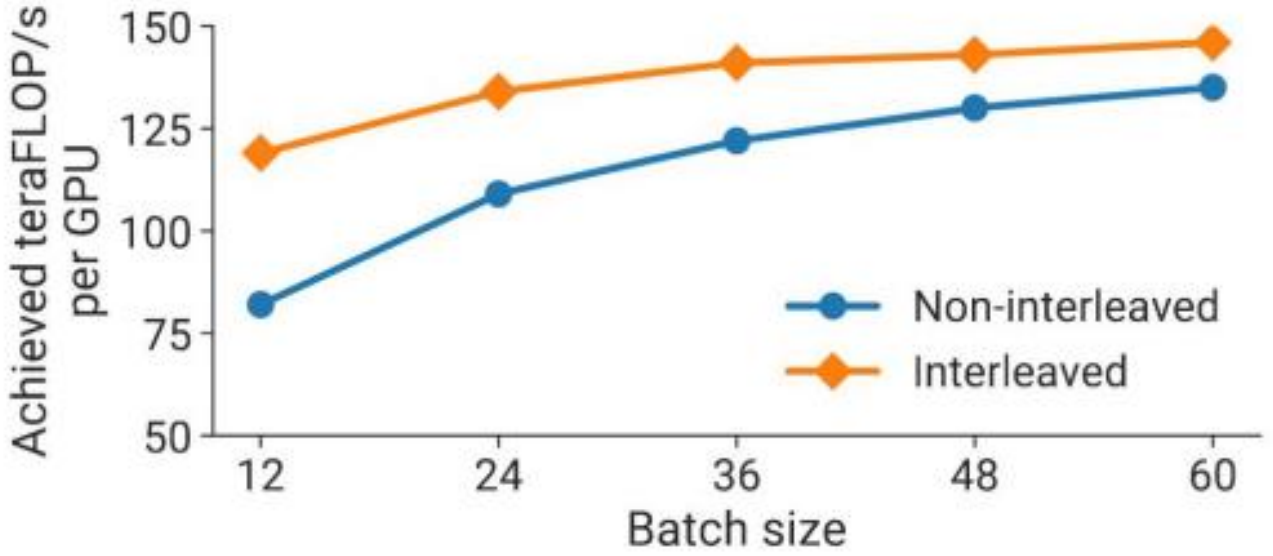
Evaluation - Pipeline Parallelism

- Weak Scaling - increase the #layers while increasing PP size
- Higher batch size scales better (p-1)/m



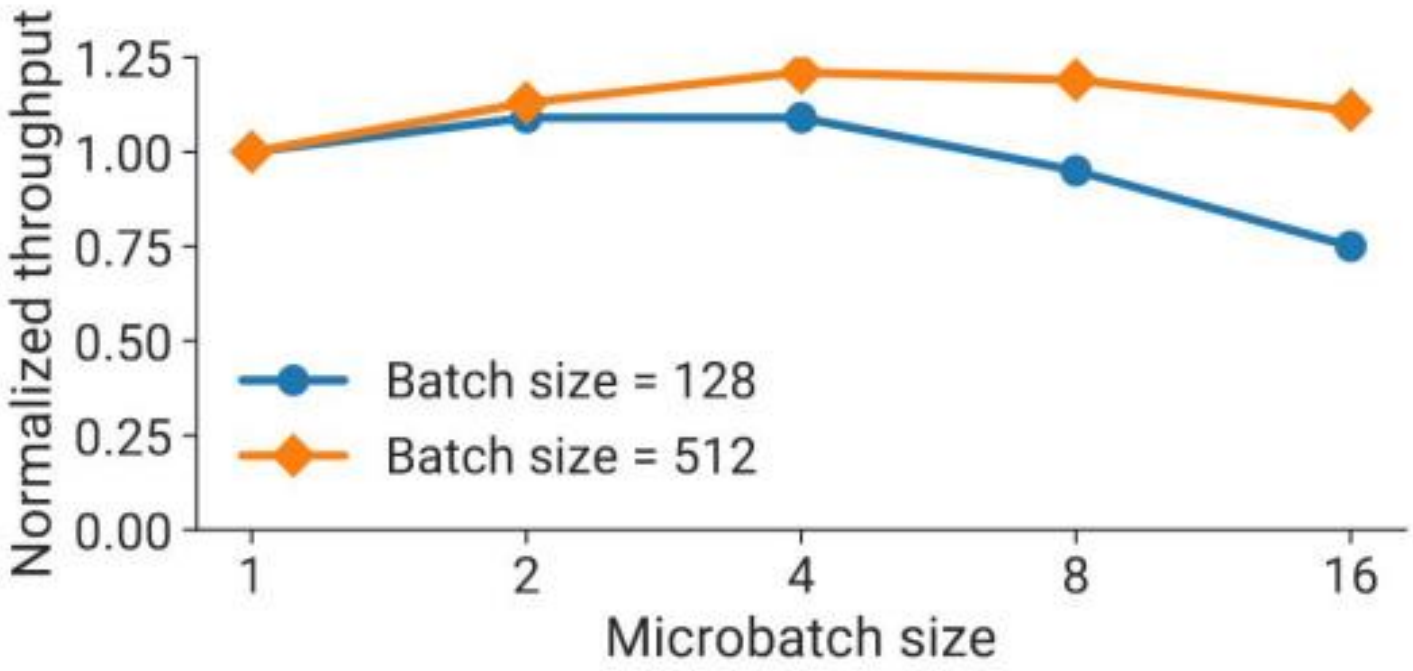
Evaluation - Pipeline Parallelism

- Interleaved schedule with scatter/gather optimization has higher throughput
  - The gap closes as the batch size increases
    - Bubble size decreases when batch size increases (i.e., more micro-batches)
    - Interleaved schedule features more communication cost per sample



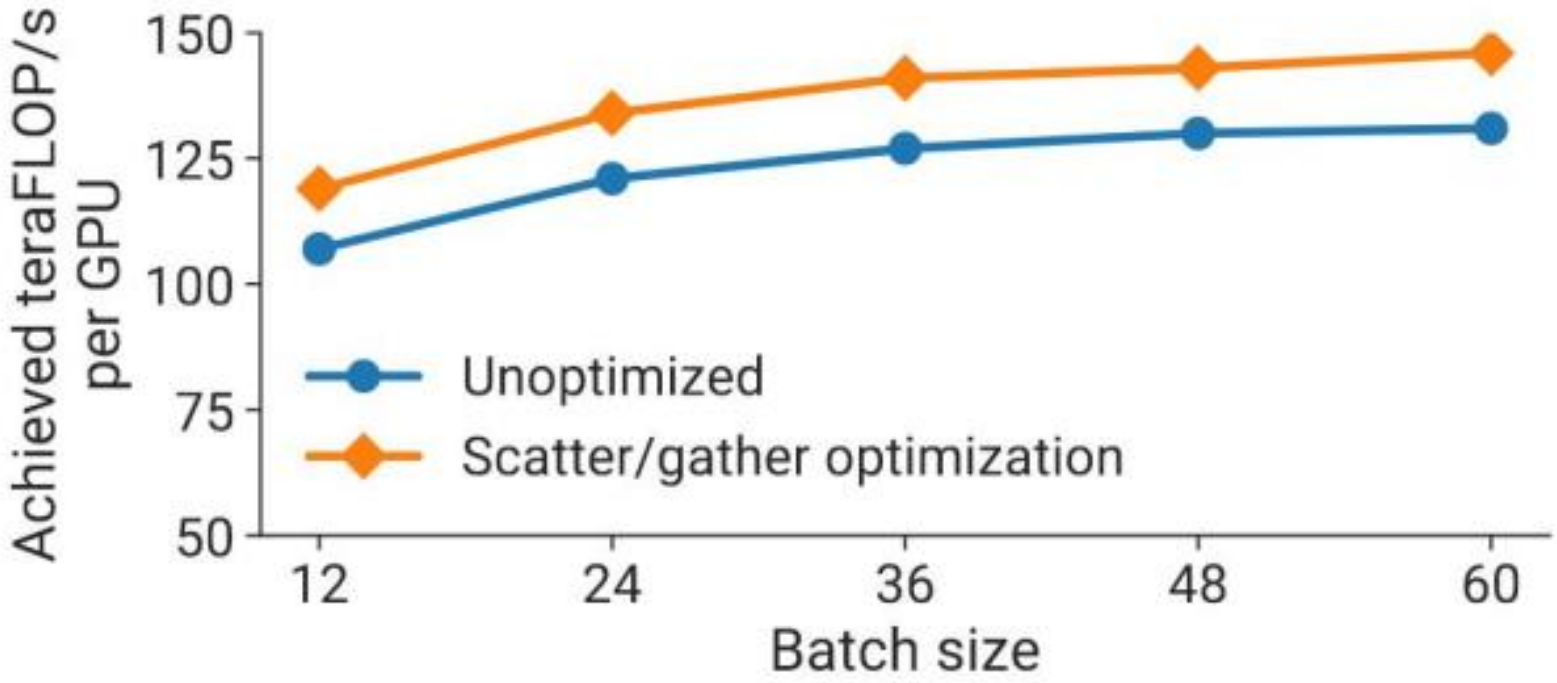
Evaluation - Selection of Microbatch size

- Optimal microbatch size is **model dependent**
  - Arithmetic intensity
  - Pipeline bubble size



Evaluation - Scatter-gather optimization

- GPT model with 175 billion parameters using 96 A100 GPUs
- Up to 11% in throughput
  - Large batch size with interleaved schedules
  - Reduce cross-node communication cost



Evaluation - End-to-end Performance

- Superlinear scaling of throughput
  - Per-GPU utilization improves as the model get larger
  - Communication overhead is not significant

Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model-parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

Evaluation - End-to-end Performance

- Estimated Training Time
  - T: number of tokens
  - P: number of parameters
  - n: number of GPUs
  - X: throughput
  - E.g. GPT3

$$\text{End-to-end training time} \approx \frac{8TP}{nX}$$

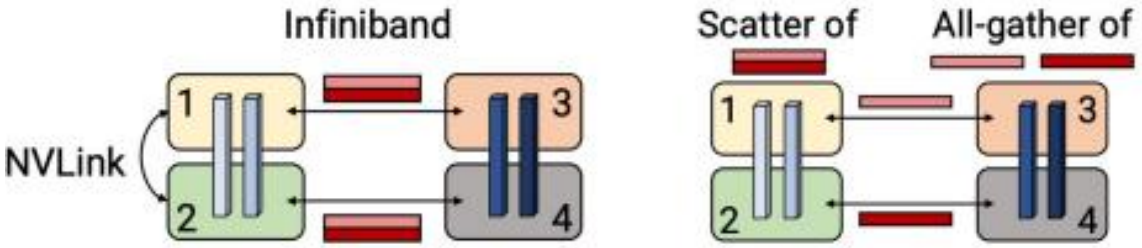
T (billion)	P (billion)	n	X (teraFLOPs/s per GPU)	#Days
300	175	1024	140	34
1000	450	3072	163	84

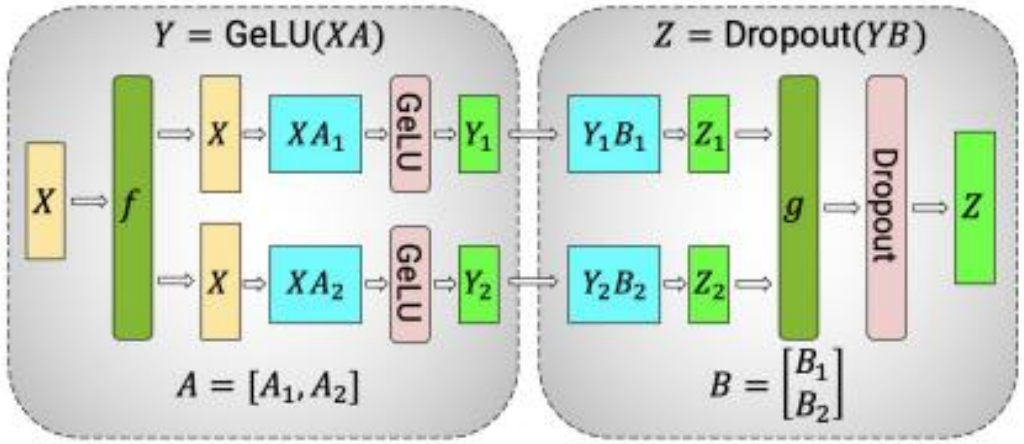
288 years with a single V100 NVIDIA GPU

Questions?

Scatter/gather Communication Optimization

- Scatter/gather optimization as an extension to the Megatron-LM
  - This reduced pipeline bubble size does not come for free
  - The output of each transformer layer is replicated (after g in MLP block)
  - They are sending and receiving the exact same set of tensors
  - Split the sending message to equal size of chunk and perform an all-gather on receivers





(a) MLP.