CS 498: Machine Learning System
Spring 2026

Minjia Zhang

The Grainger College of Engineering

Today

Pipeline Parallelism

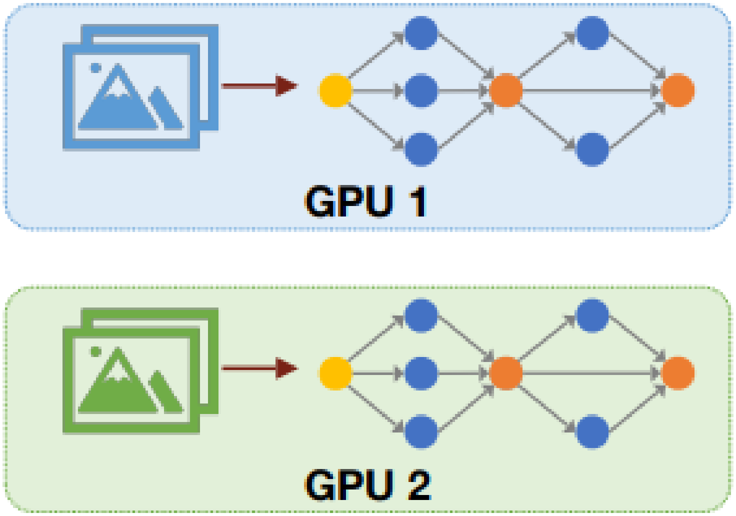
Multi-Dimensional Parallelism

First assignment (due in two weeks Feb 25 EOD)

Learning Objective

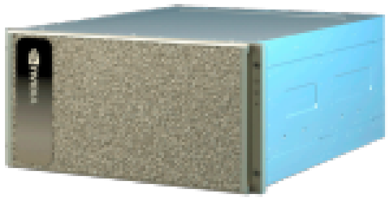
- **Analyze pipeline execution and quantify bubble overhead**
- **Understand differences in pipeline schedules**

Data Parallelism Cannot Train Large Models



175B * 16 Bits
= 350GB Memory

>

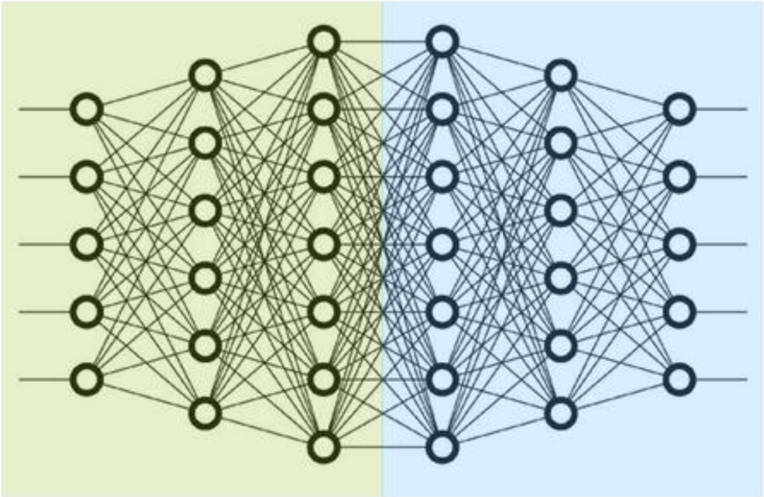


Nvidia H100 94 GB

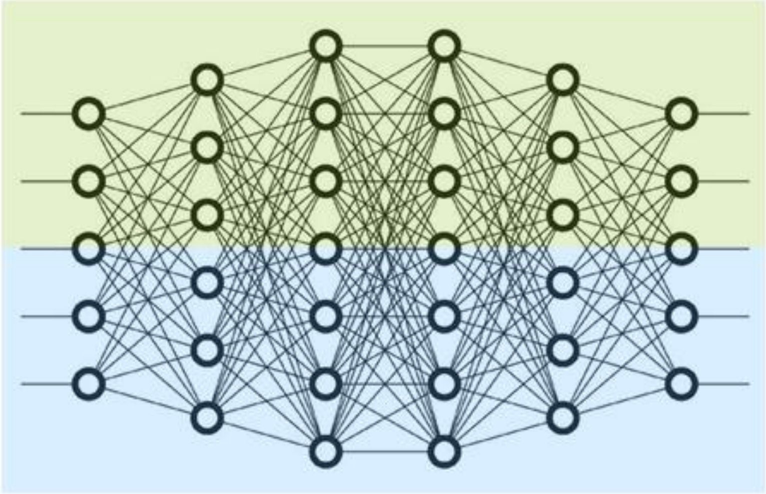
Even the best GPU **CANNOT** fit the model into memory!

Model Parallelism

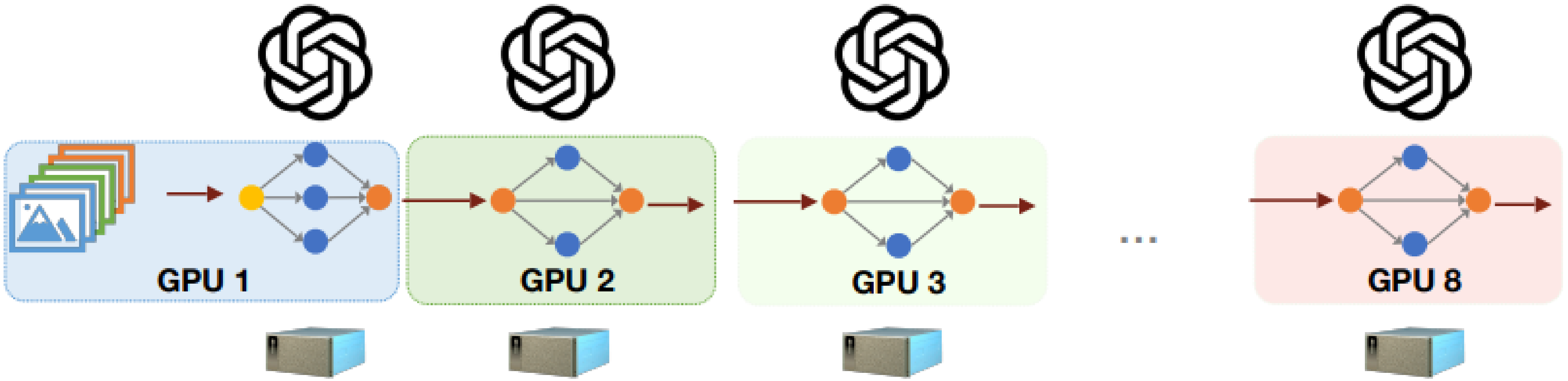
Inter-layer (Pipeline) parallelism
- Split sets of layers across multiple devices
- Layer 0, 1, 2 and layer 3, 4, 5 are on different devices



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



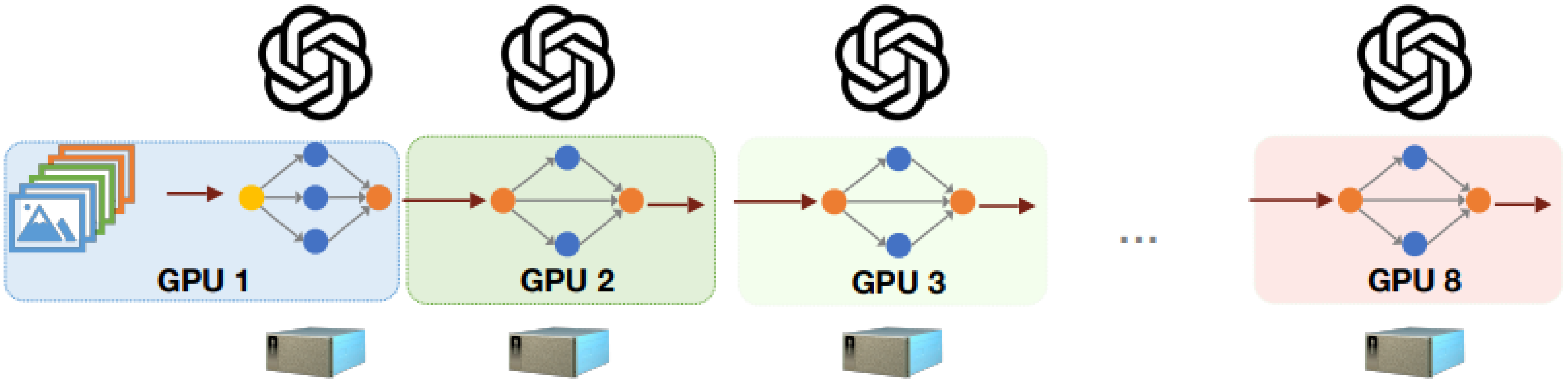
Inter-Layer Model Parallelism



350GB / 8 cards = 43.75G < 80G

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

Inter-Layer Model Parallelism

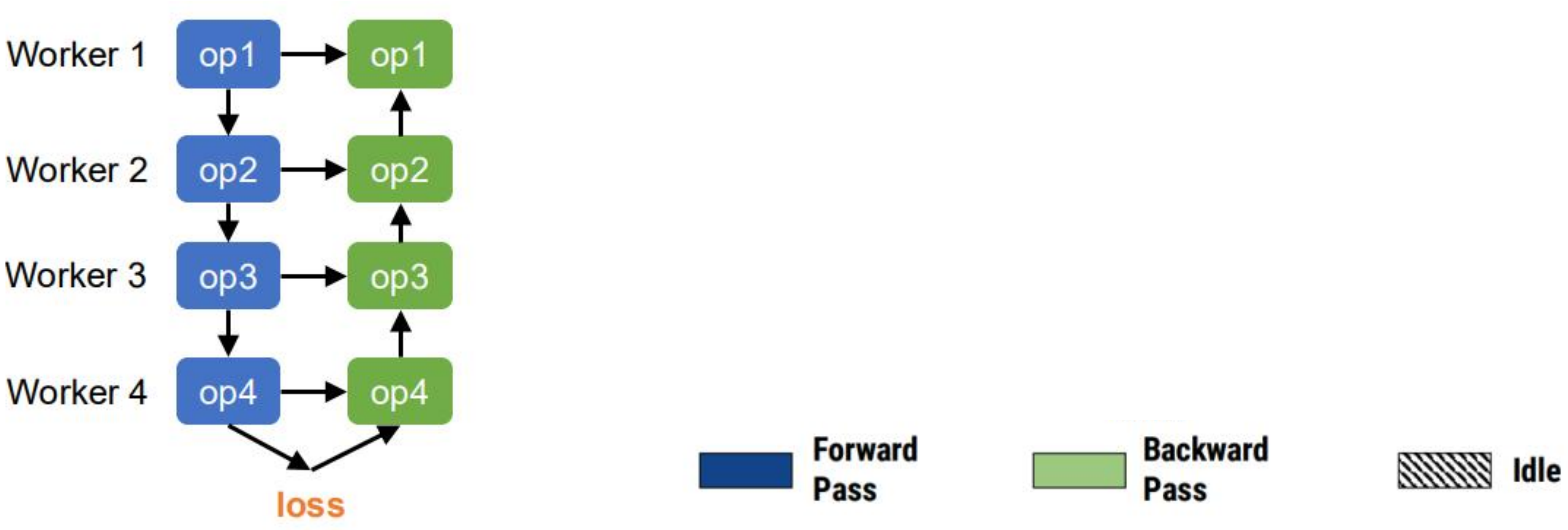


350GB / 8 cards = 43.75G < 80G

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?

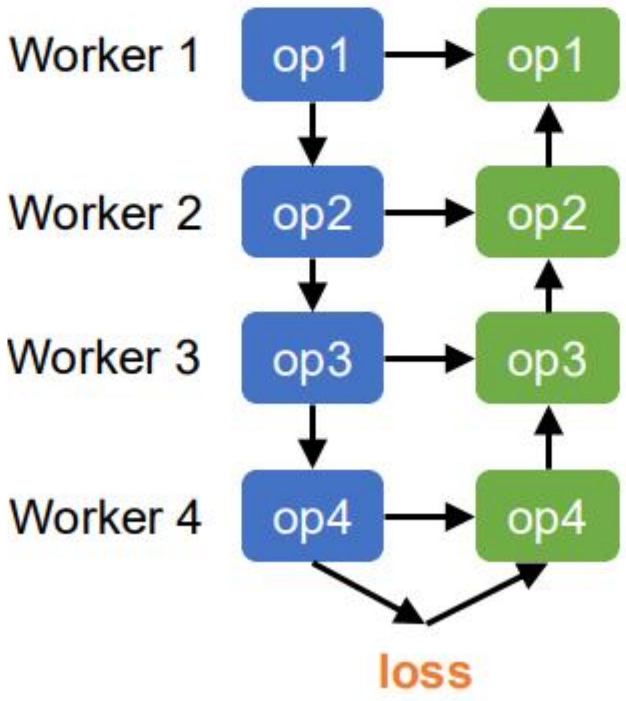
Simple Inter-Layer Parallelism

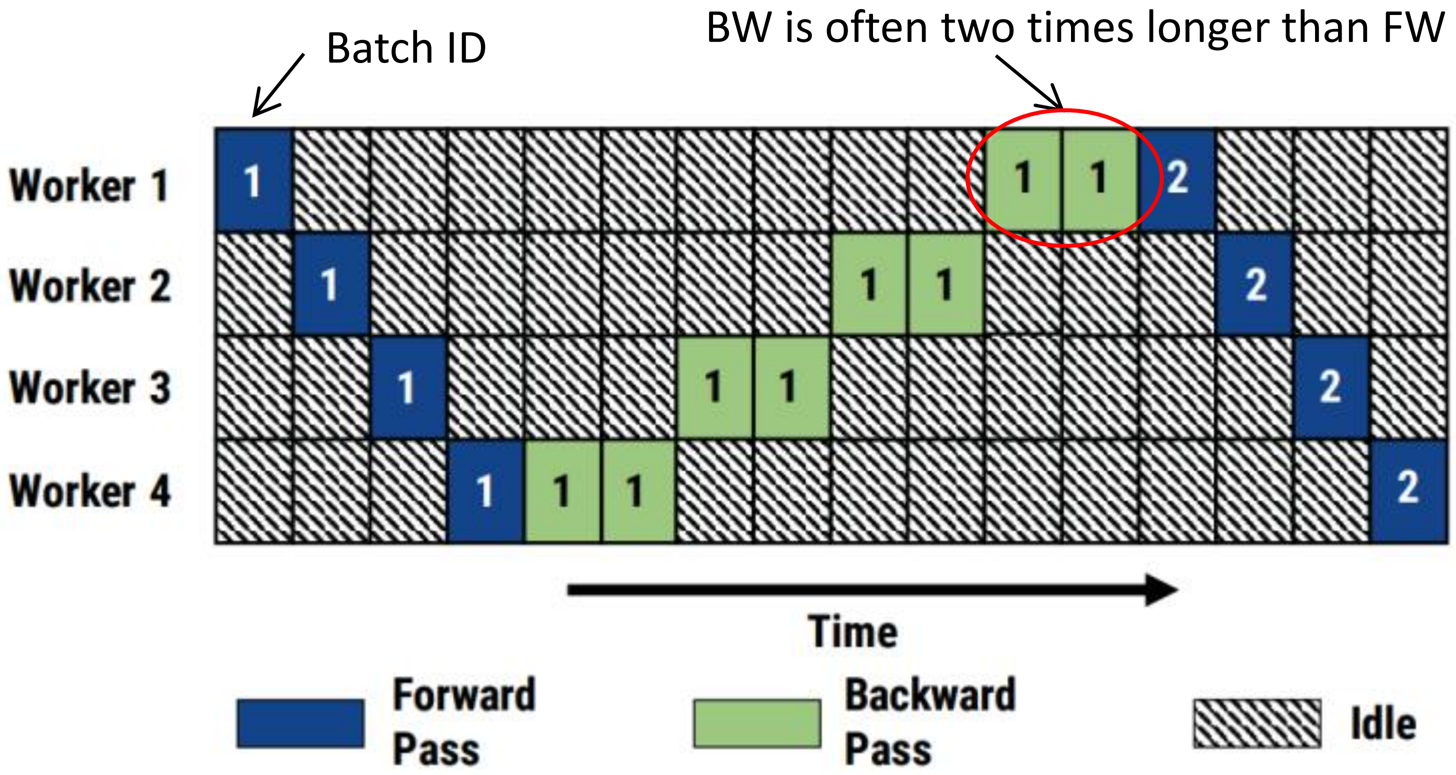


DNN training involves a bi-directional execution

- The forward pass for a minibatch starts at the input layer
- The backward pass ends at the input layer

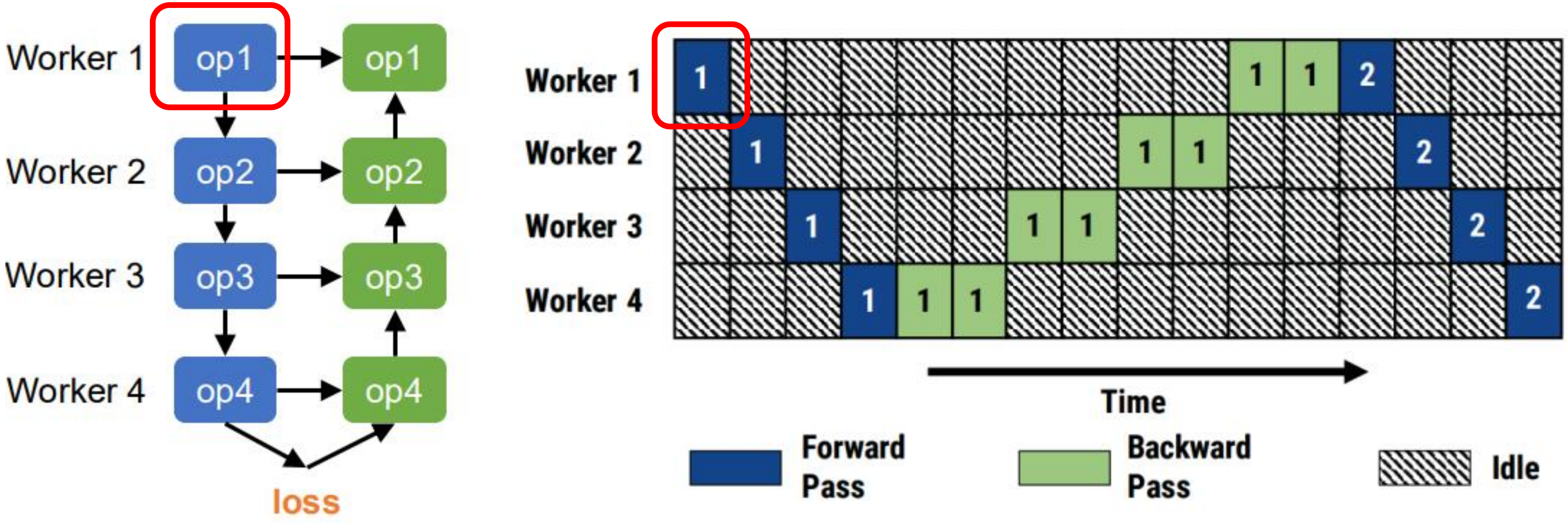
Simple Inter-Layer Parallelism



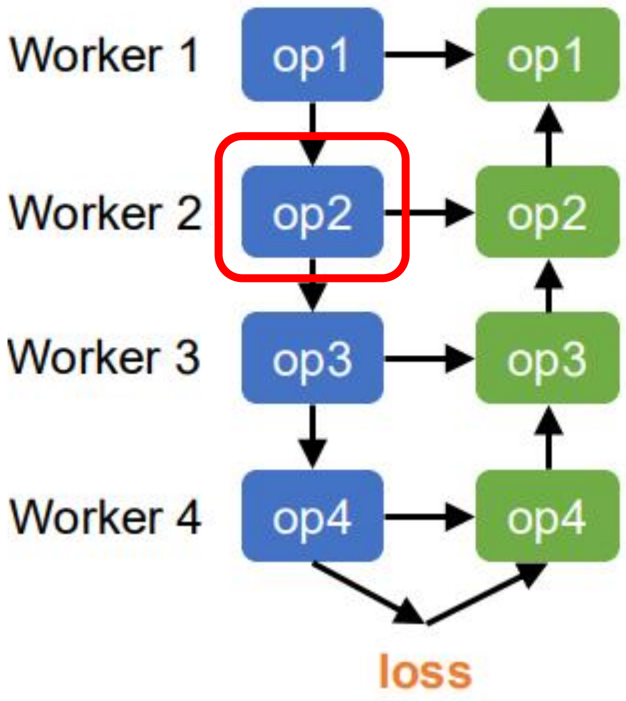


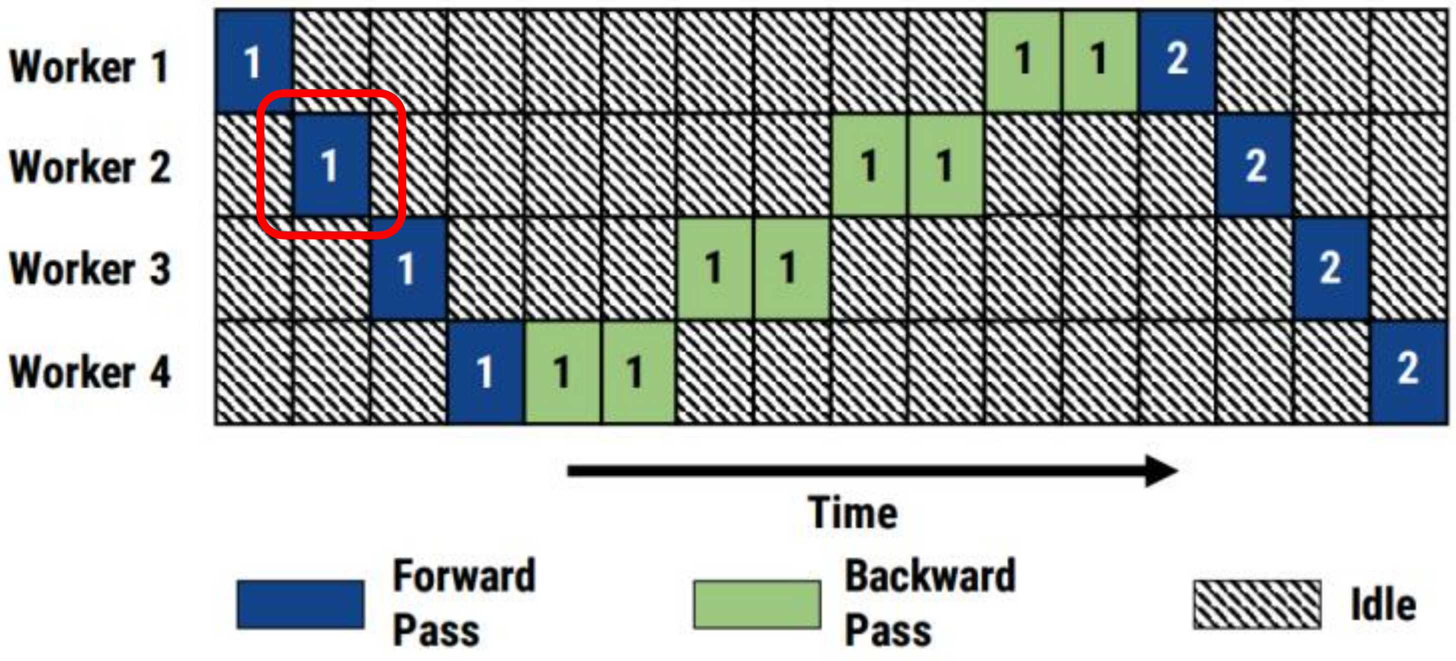
DNN training involves a bi-directional execution
- The forward pass for a minibatch starts at the input layer
- The backward pass ends at the input layer

Simple Inter-Layer Parallelism

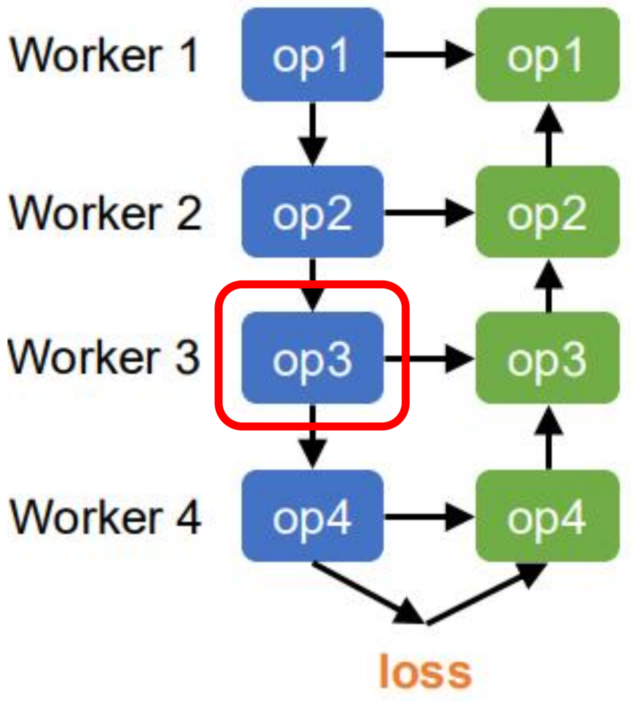


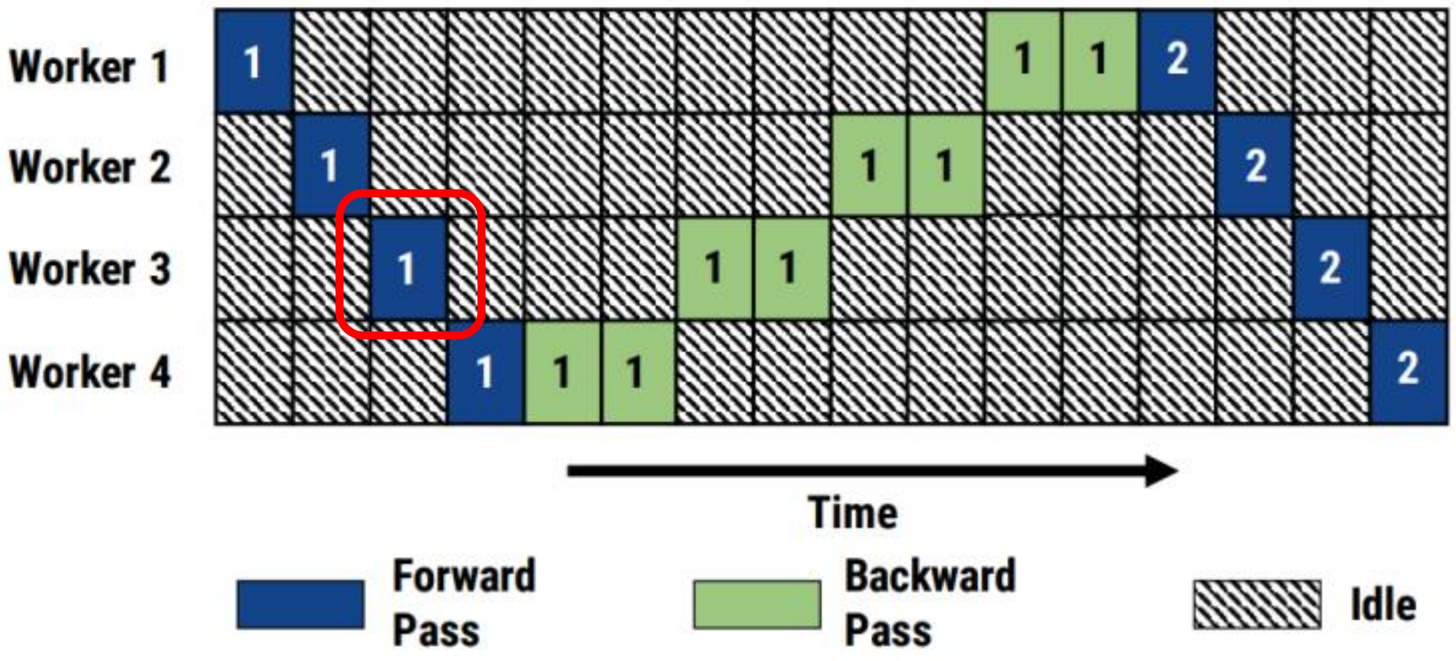
Simple Inter-Layer Parallelism



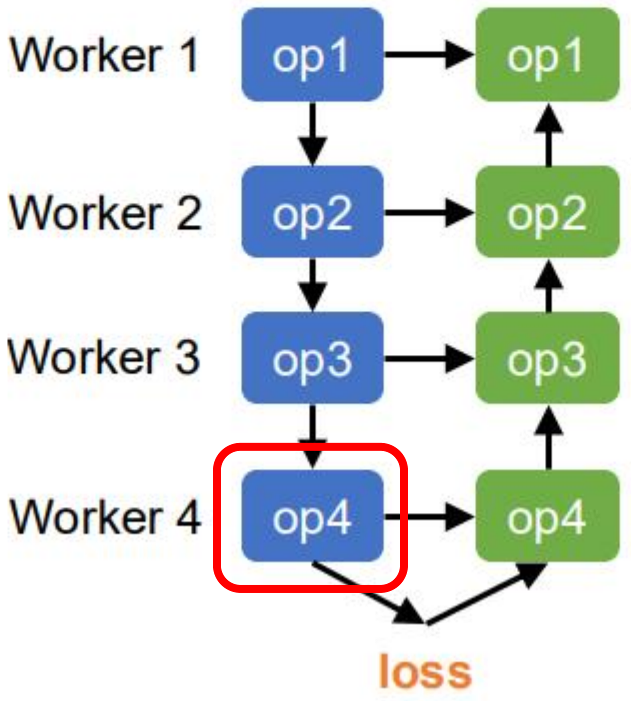


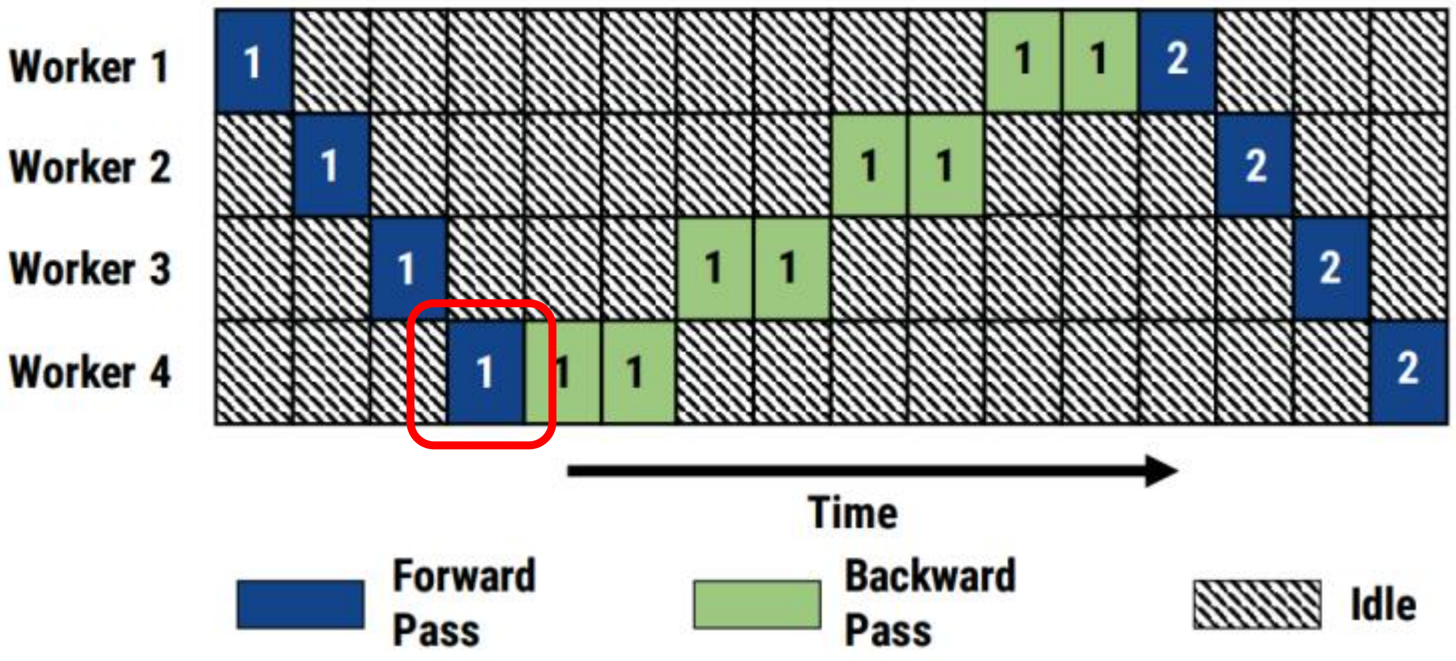
Simple Inter-Layer Parallelism



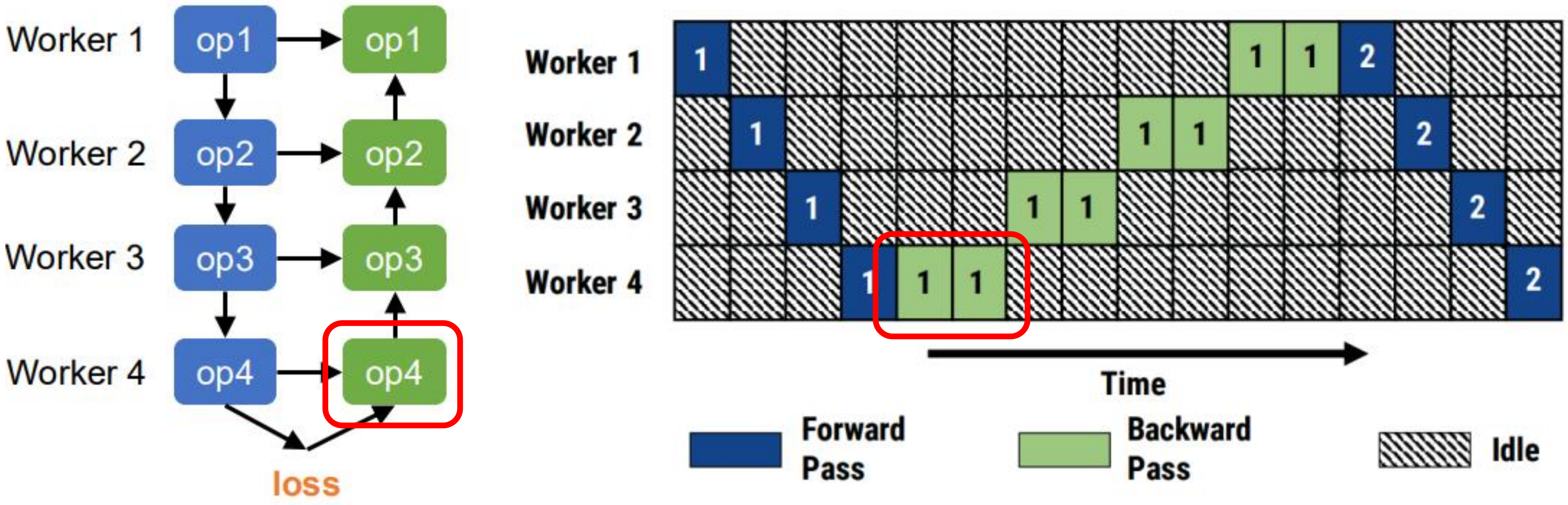


Simple Inter-Layer Parallelism

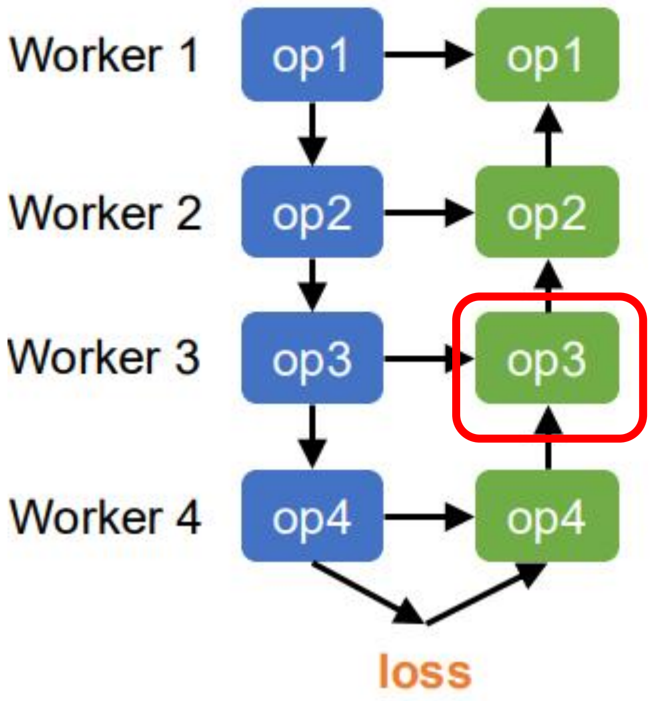


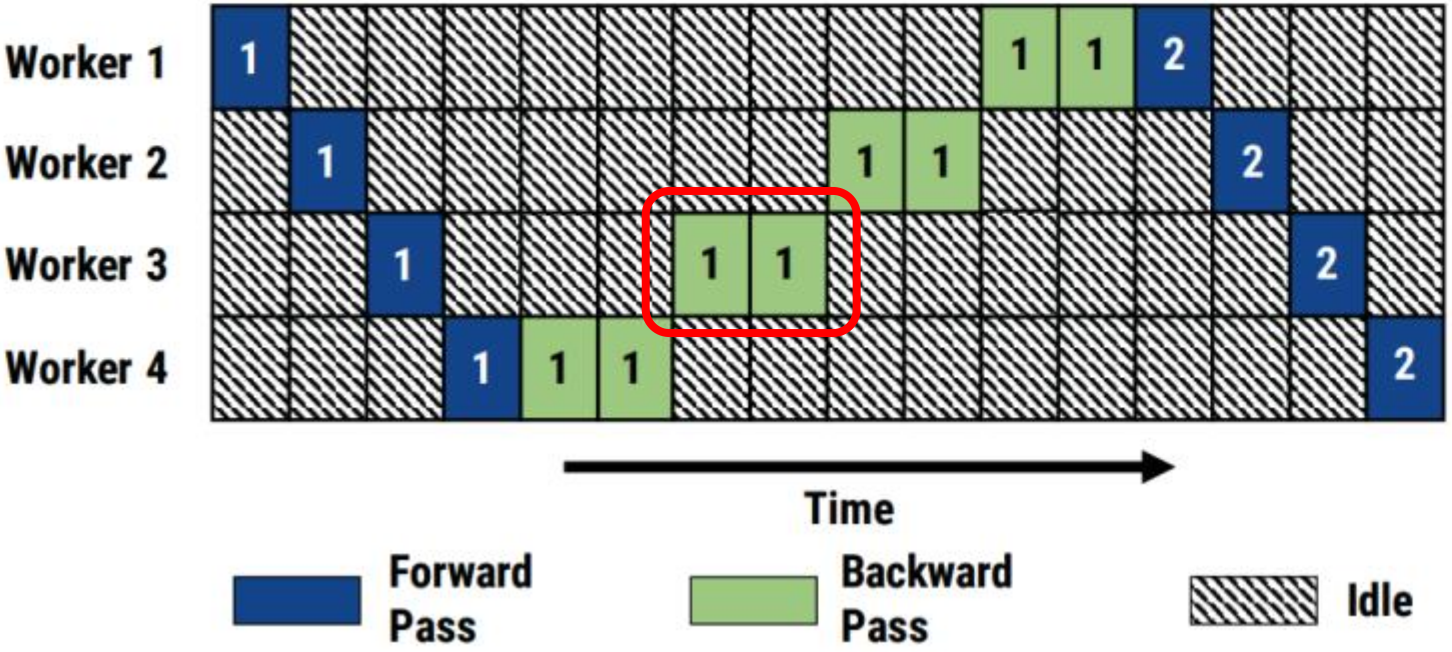


Simple Inter-Layer Parallelism

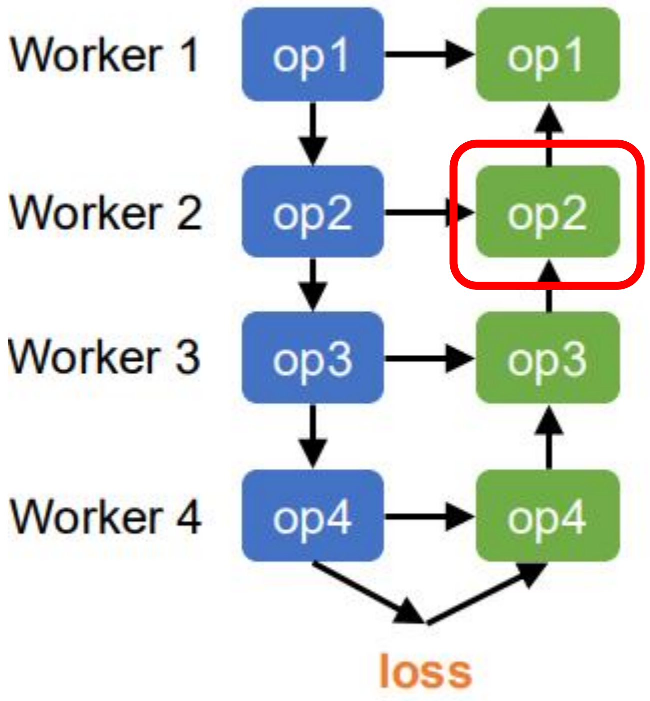


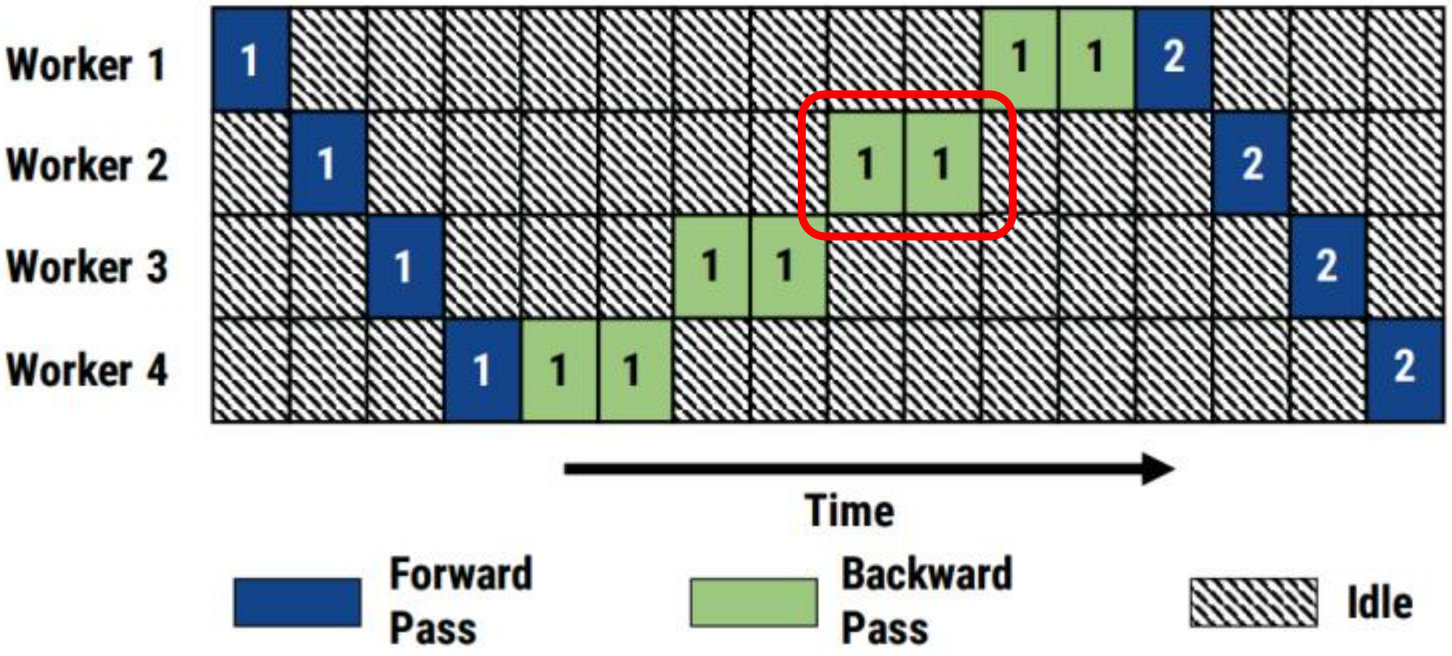
Simple Inter-Layer Parallelism



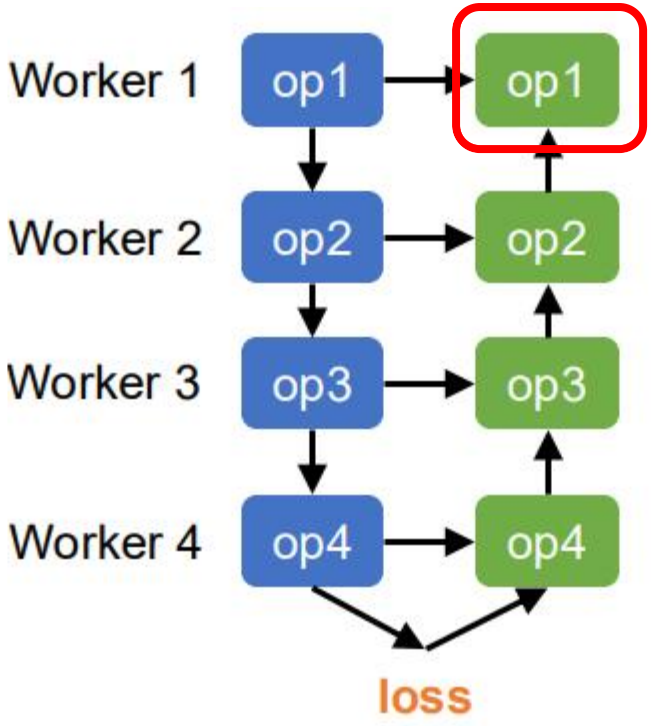


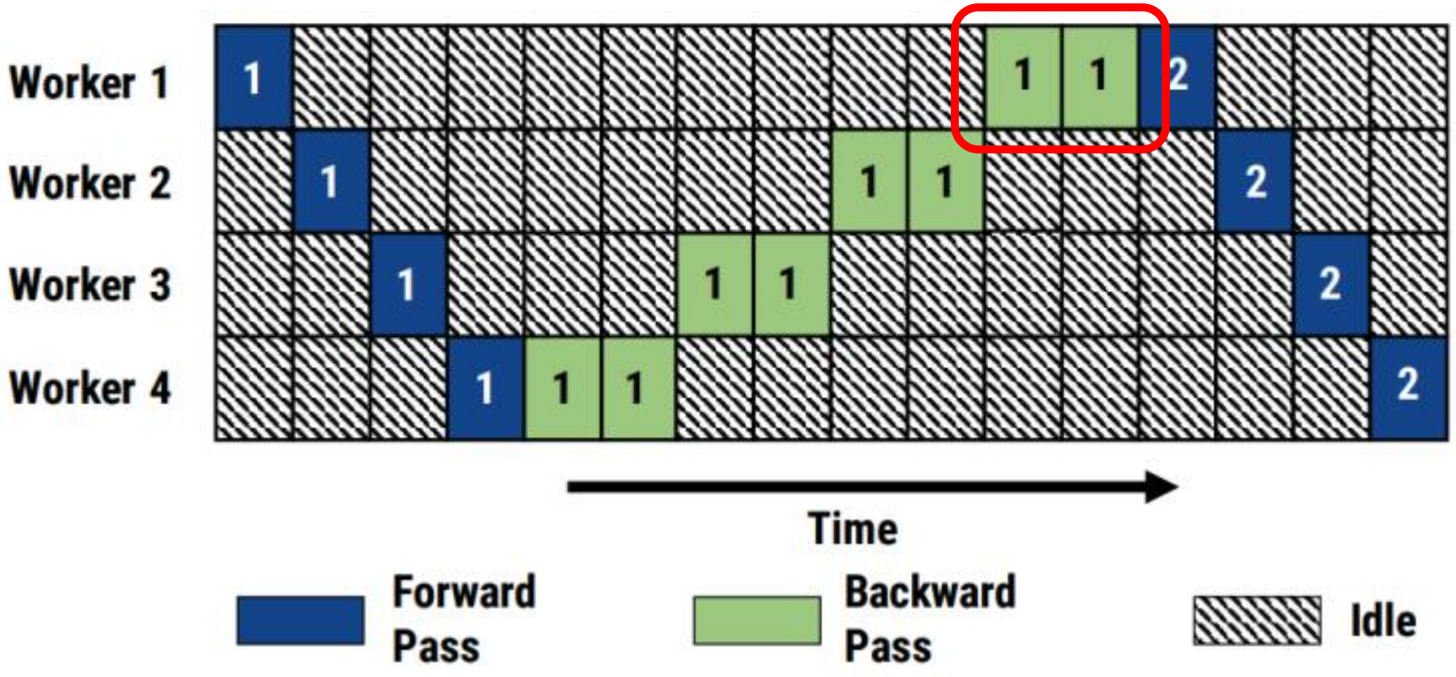
Simple Inter-Layer Parallelism



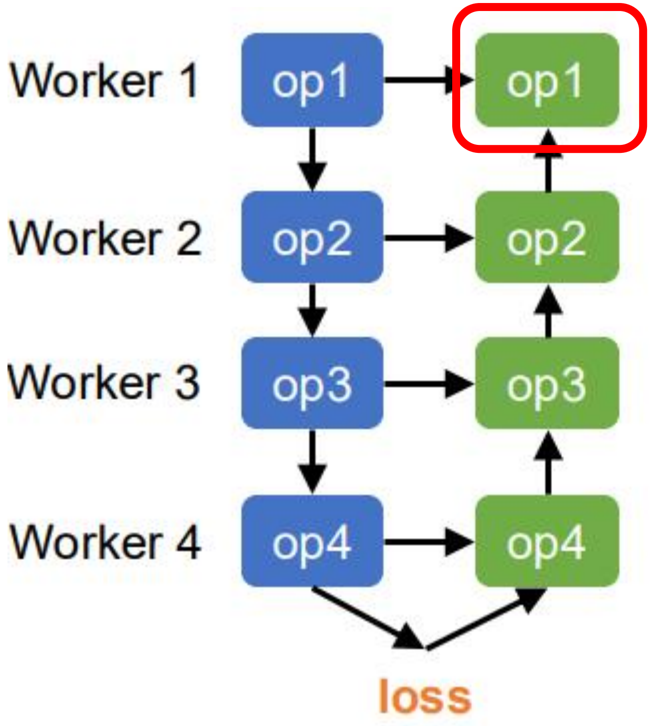


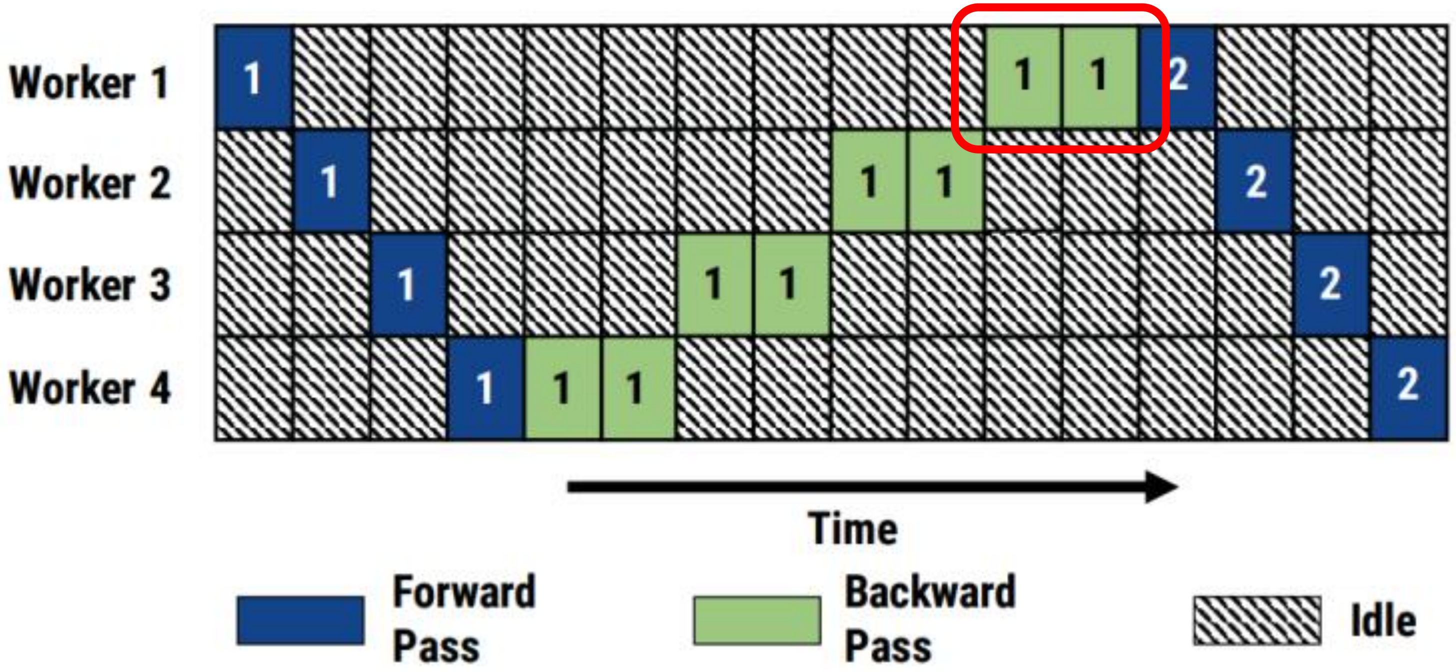
Simple Inter-Layer Parallelism





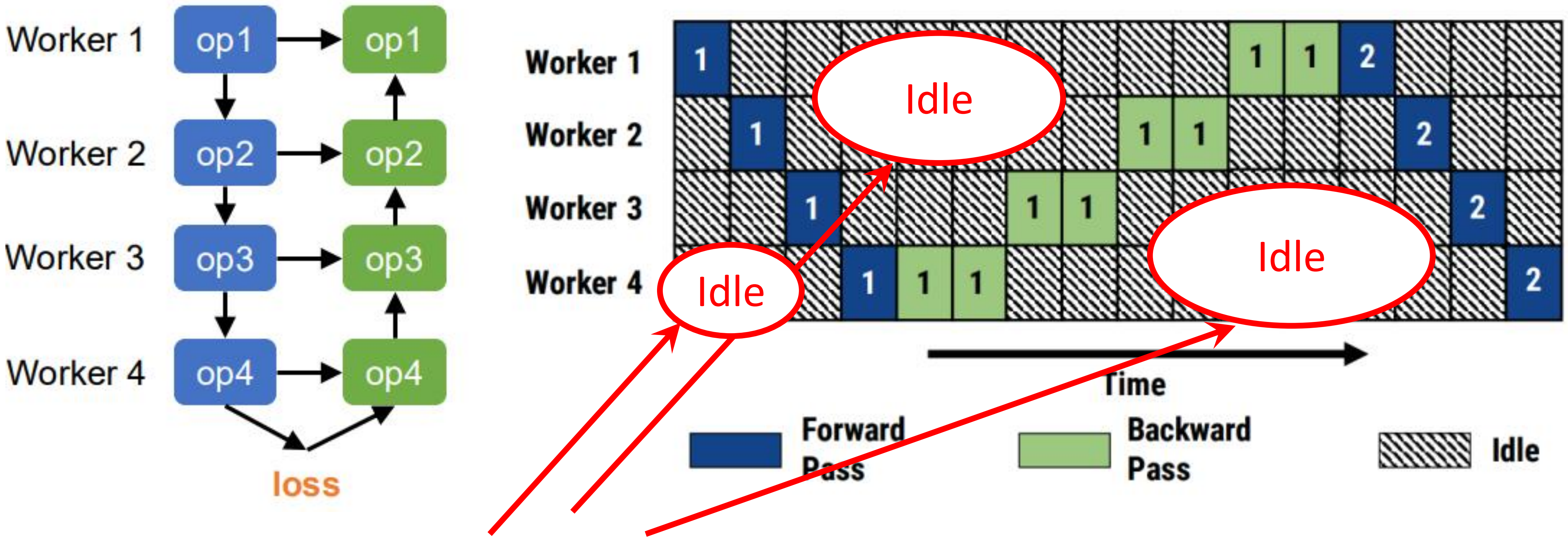
Simple Inter-Layer Parallelism





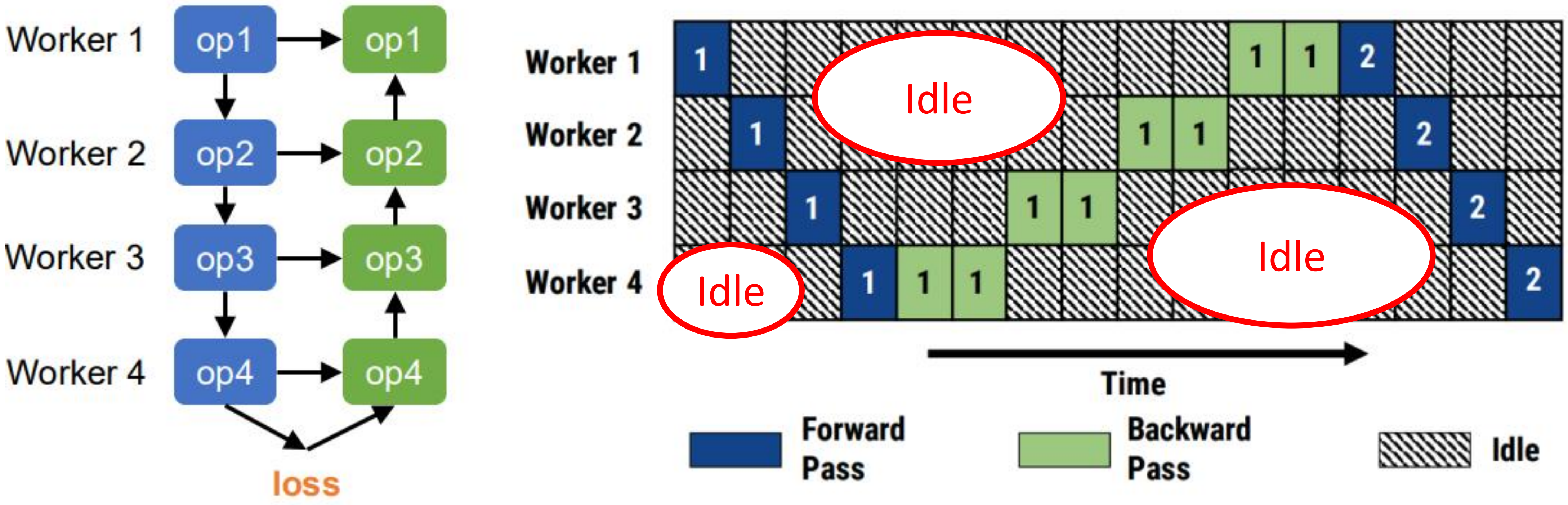
Question: What is the limitation of this execution?

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

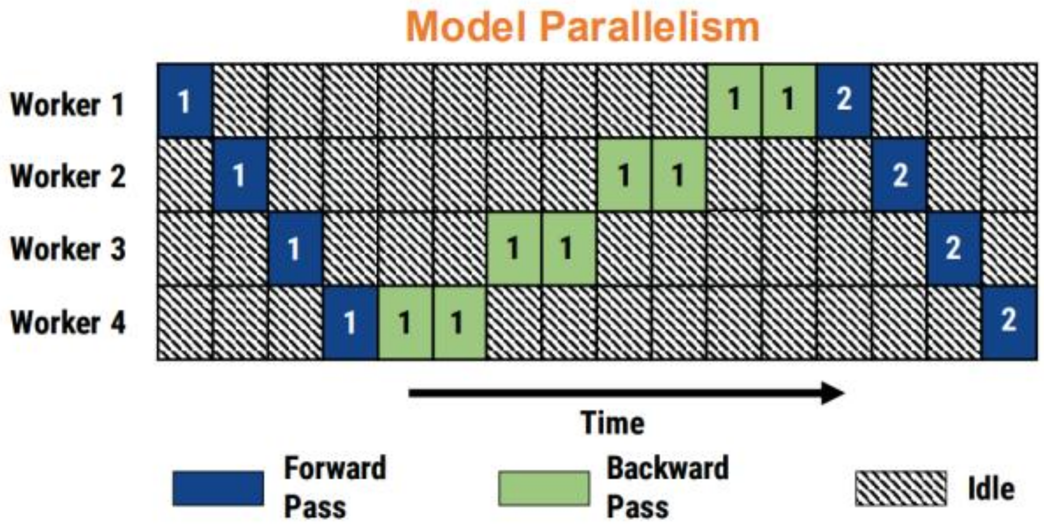


Question: How to improve the utilization and let multiple workers work simultaneously?

Pipeline Model Parallelism

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

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

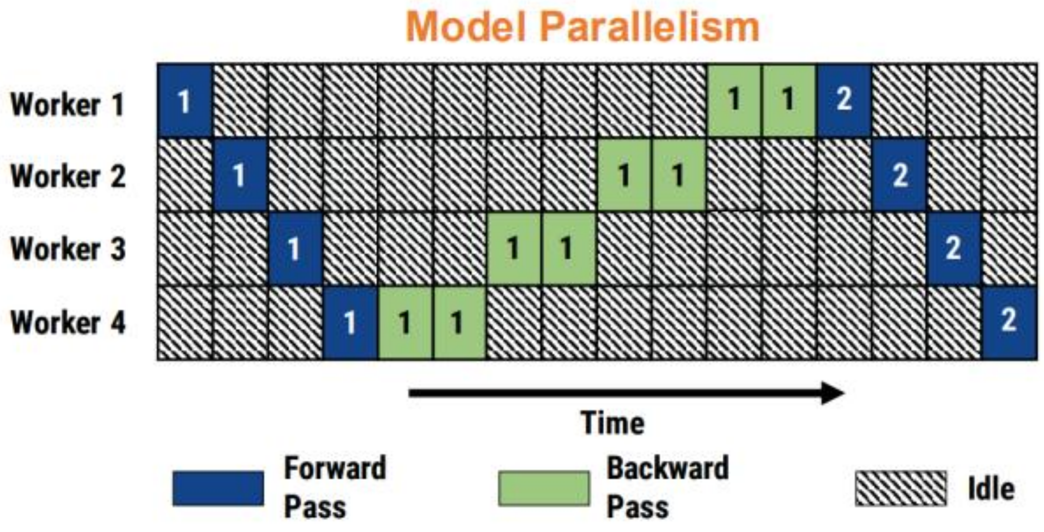


Pipeline Model Parallelism

- 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

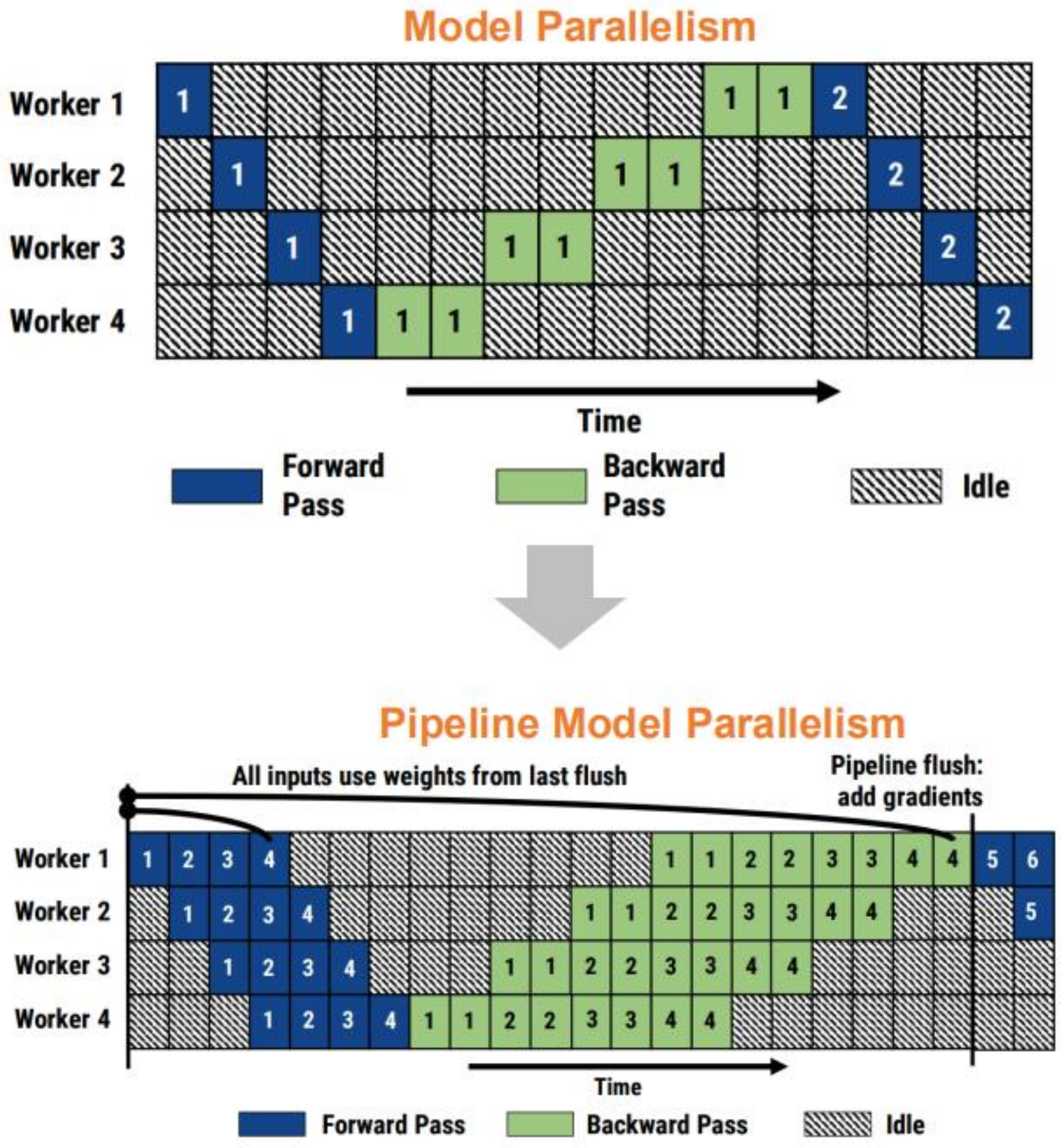


Pipeline Model Parallelism

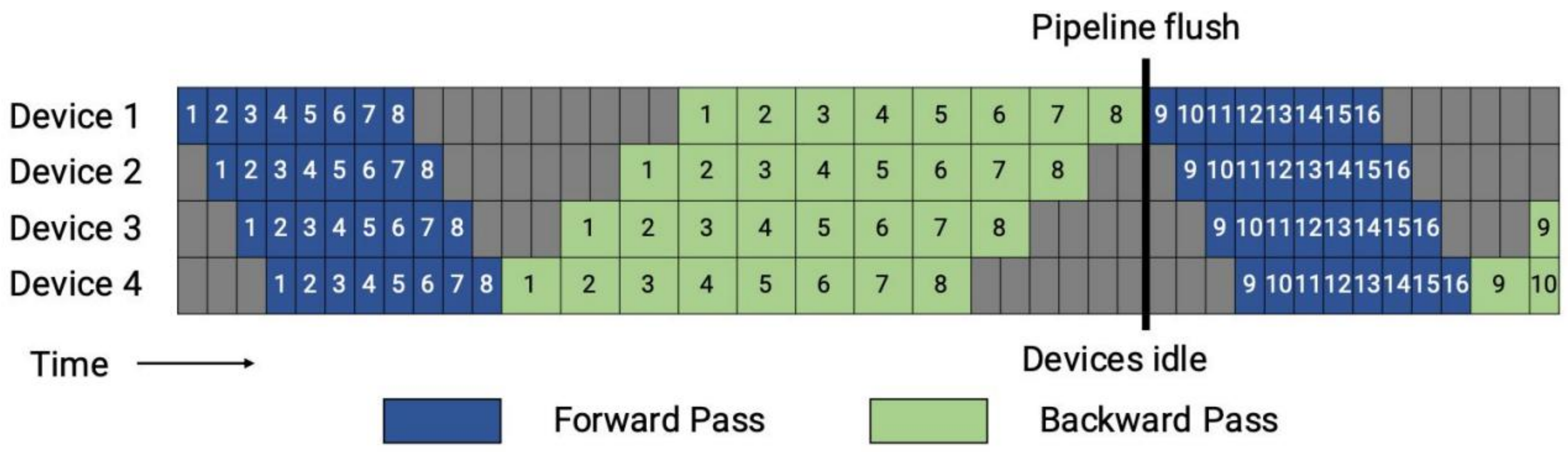
- Mini-batch: the number of samples processed in each iteration

- 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

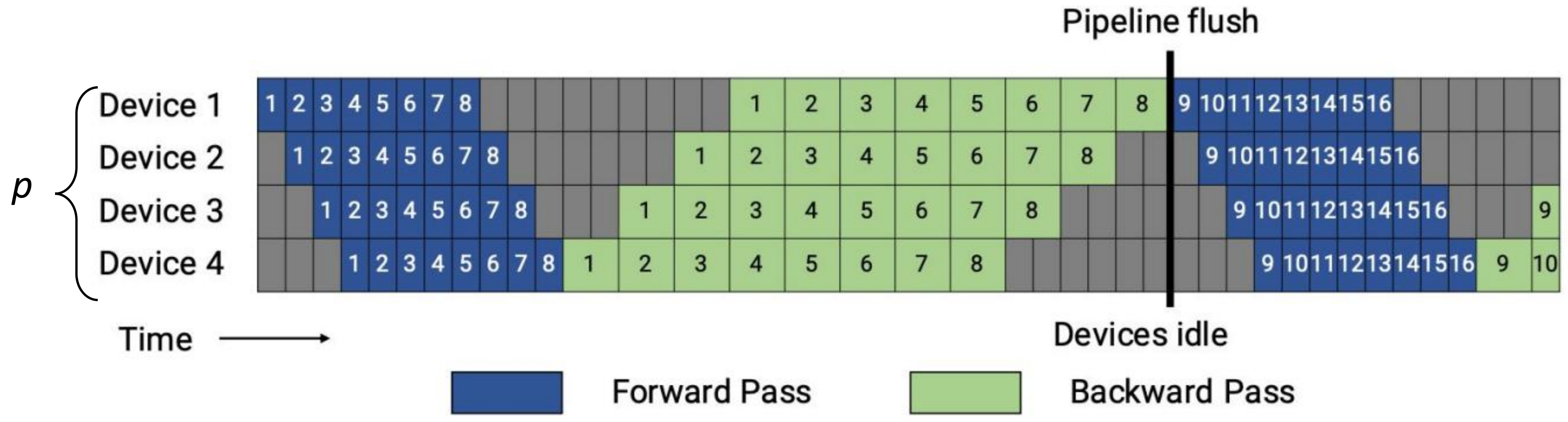


Still have bubbles in the pipeline

Question: How do we quantify bubbles?

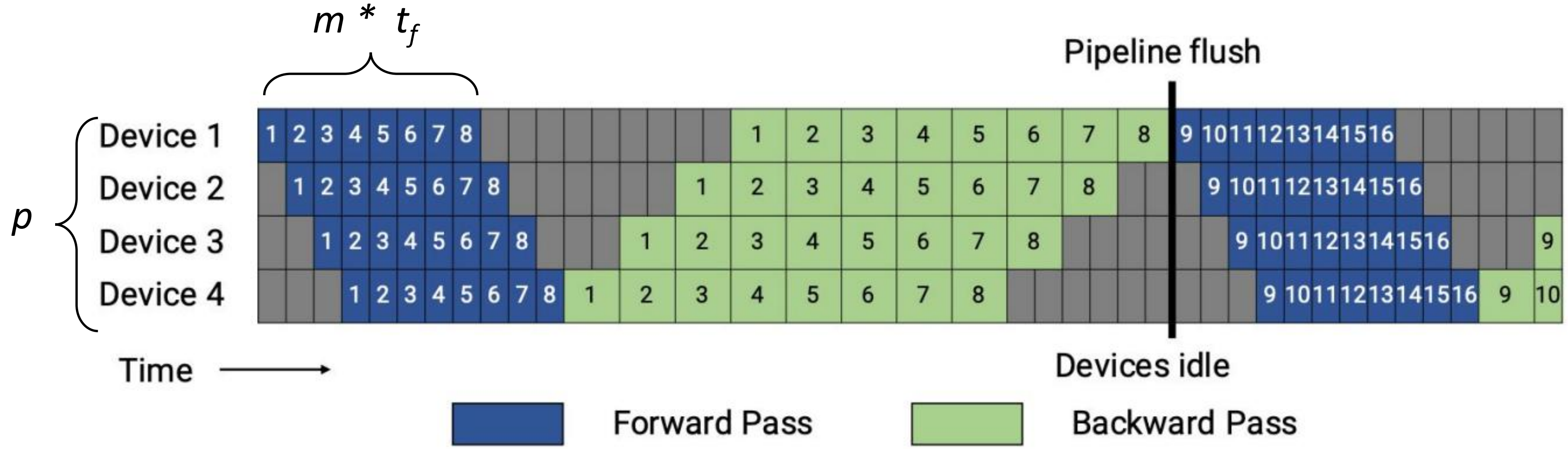
Pipeline Model Parallelism: Device Utilization

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



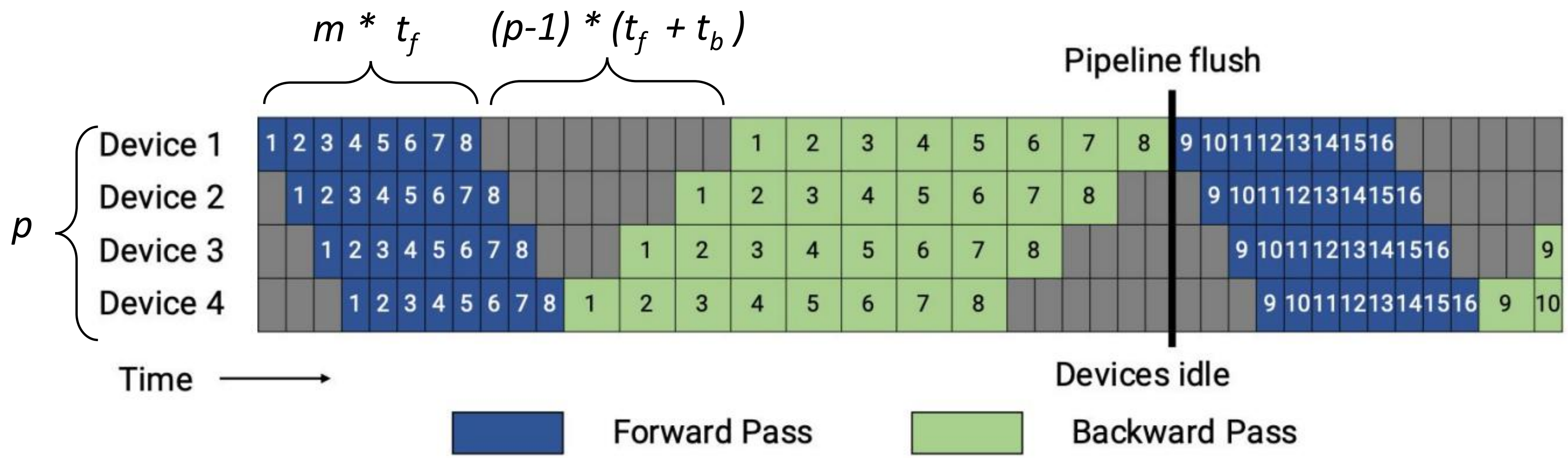
Pipeline Model Parallelism: Device Utilization

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



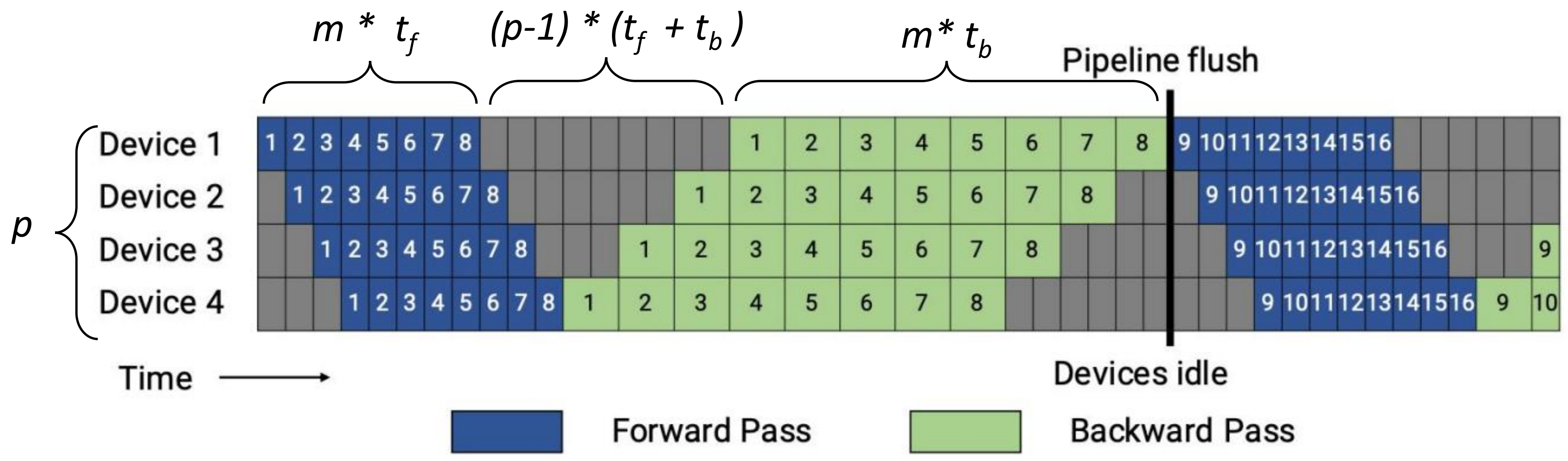
Pipeline Model Parallelism: Device Utilization

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



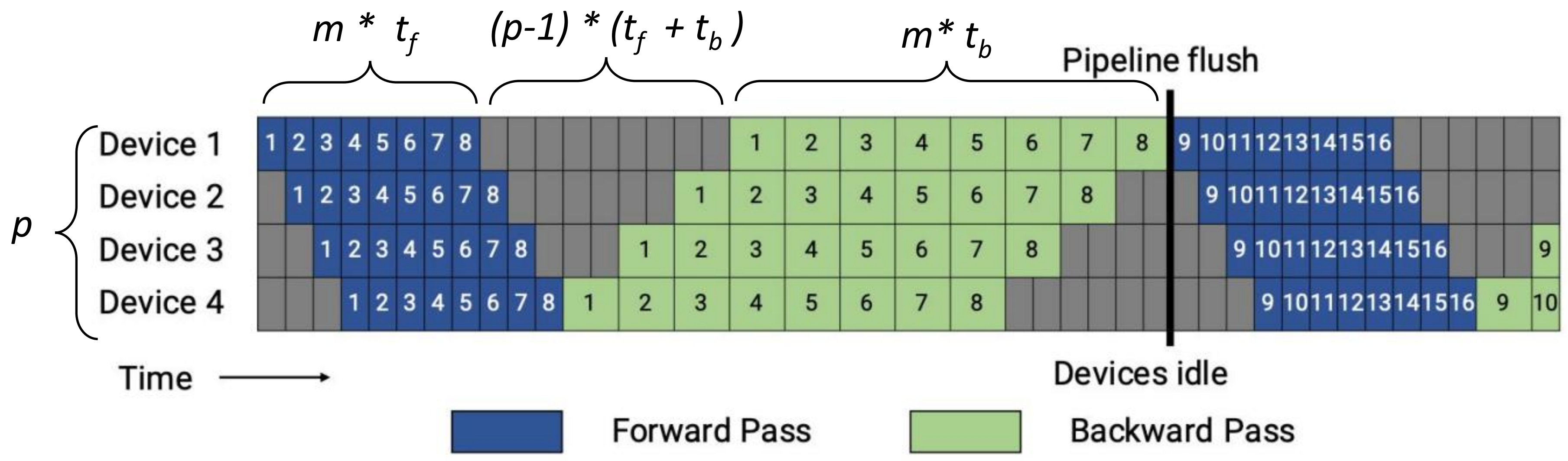
Pipeline Model Parallelism: Device Utilization

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



Pipeline Model Parallelism: Device Utilization

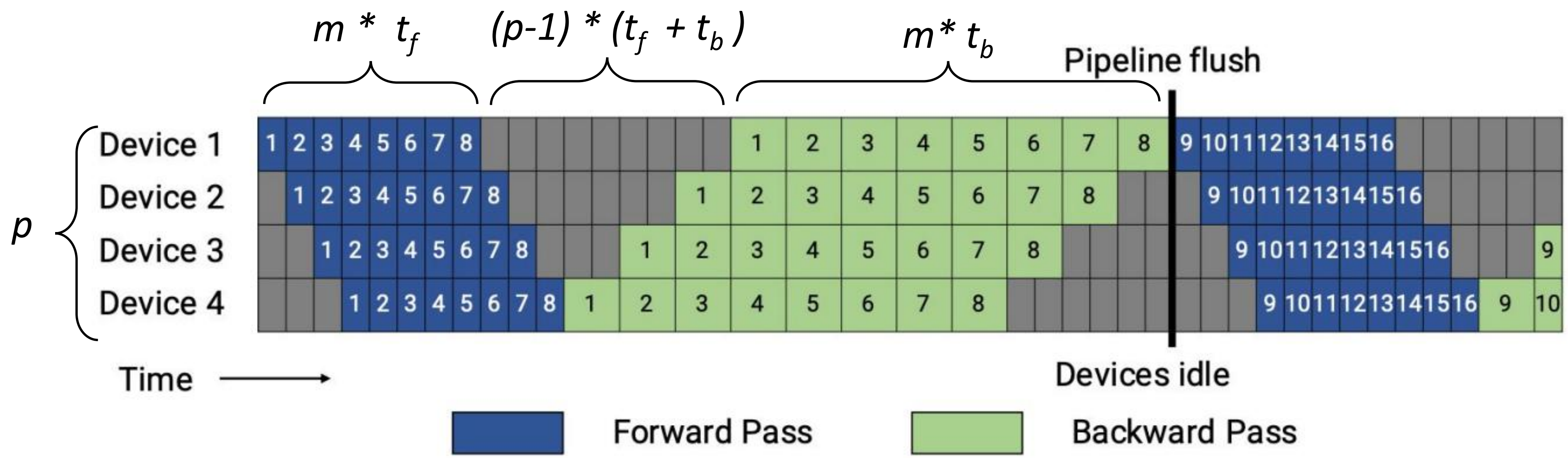
m = #microbatches (8)
p = pipeline stages (4)
t_f = time of forward
t_b = time of backward



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

Pipeline Model Parallelism: Device Utilization

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

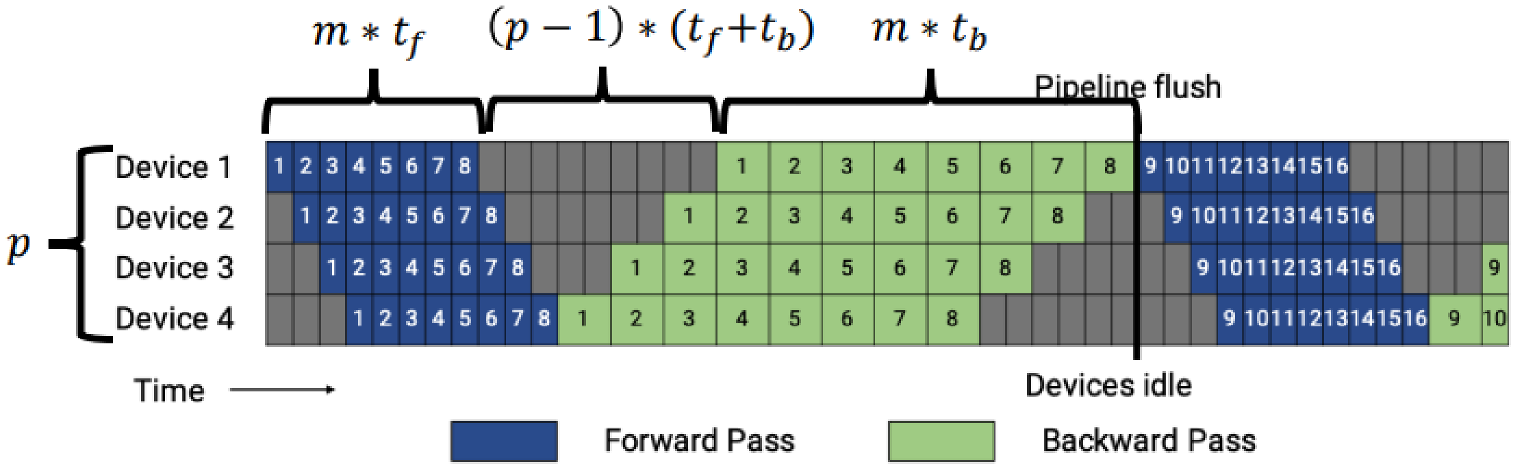


$$BubbleFraction = \frac{(p-1)*(t_f+t_b)}{m*t_f + m*t_b} = \frac{p-1}{m}$$

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



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

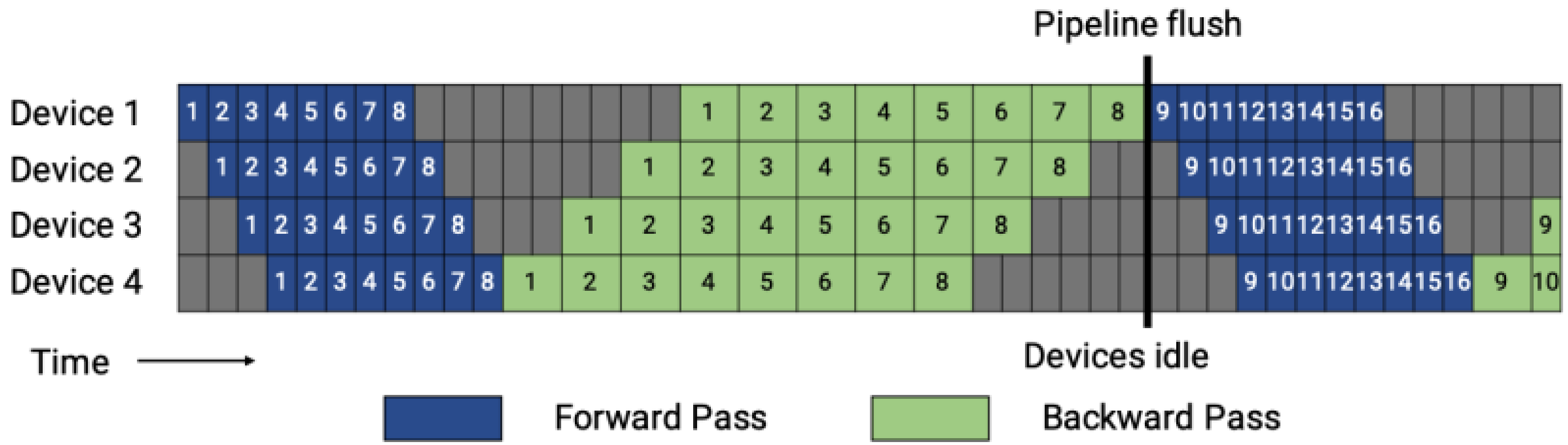
Design More Advanced Pipeline Schedule

Question: How to further optimize pipeline parallelism?

- 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

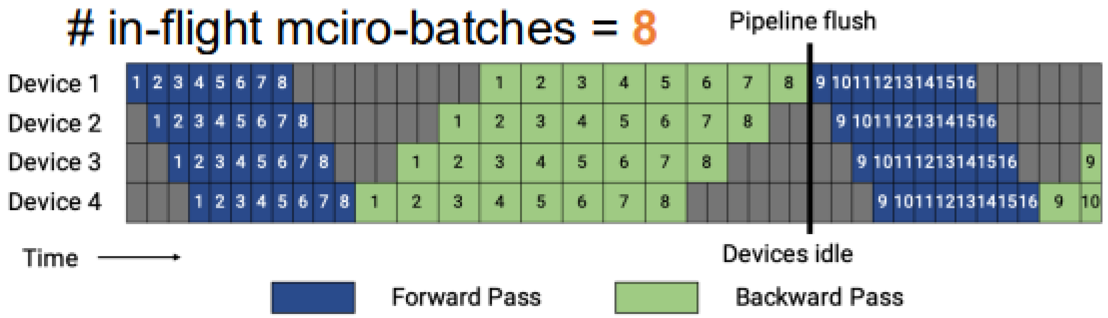
- An issue: we need to keep the intermediate activations of **all micro-batches** before back propagation



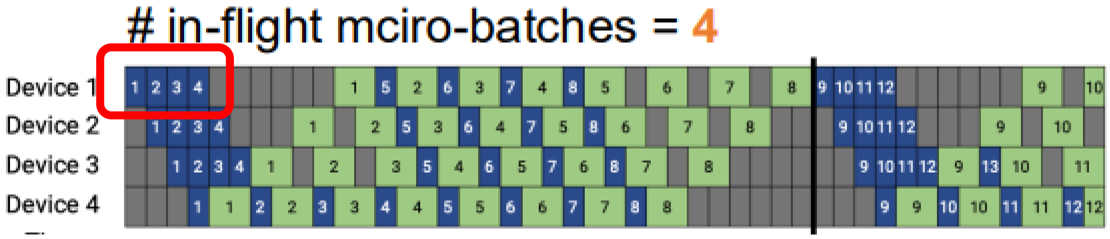
Question: Can we improve the pipeline schedule to reduce memory requirements?

Pipeline Parallelism with 1F1B Schedule

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



Pipeline parallelism with GPipe's schedule

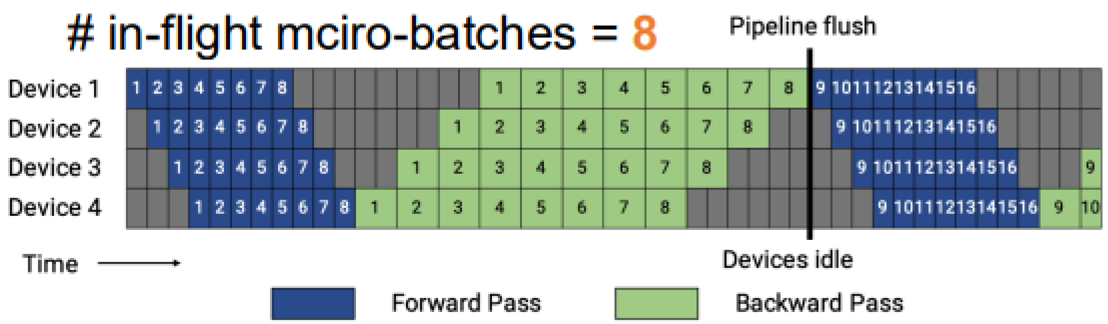


Pipeline parallelism with 1F1B schedule

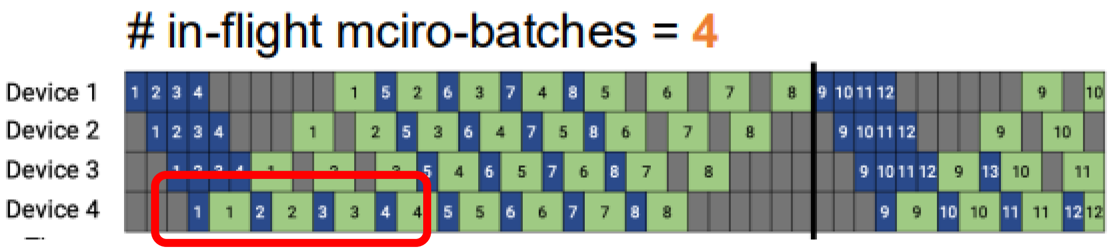
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

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



Pipeline parallelism with GPipe's schedule

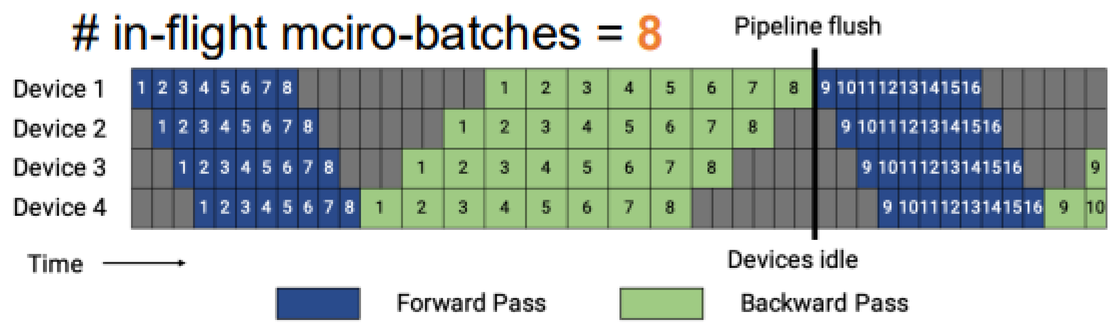


Pipeline parallelism with 1F1B schedule

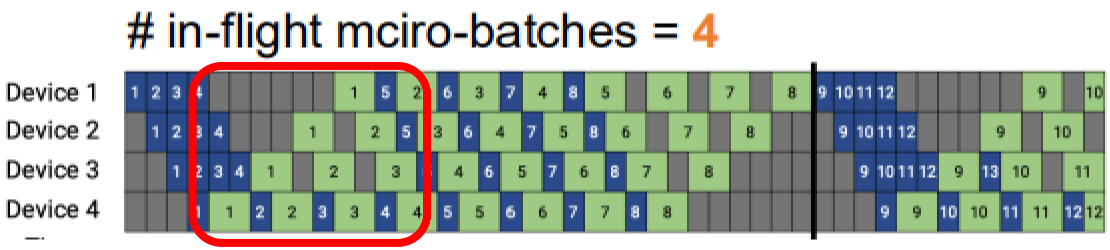
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

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



Pipeline parallelism with GPipe's schedule

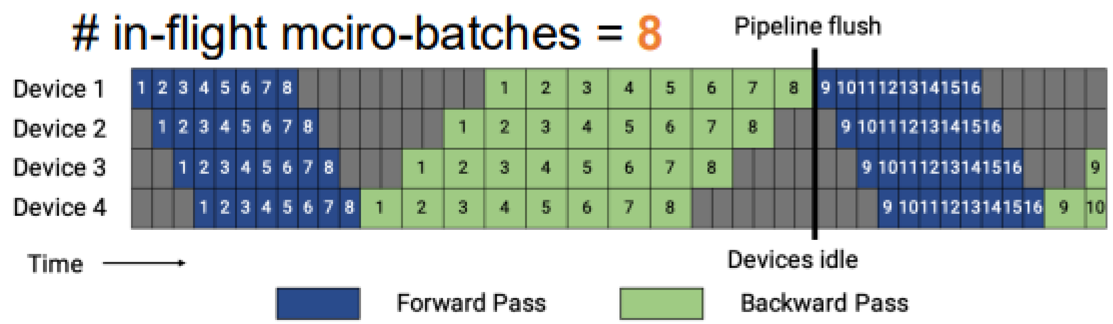


Pipeline parallelism with 1F1B schedule

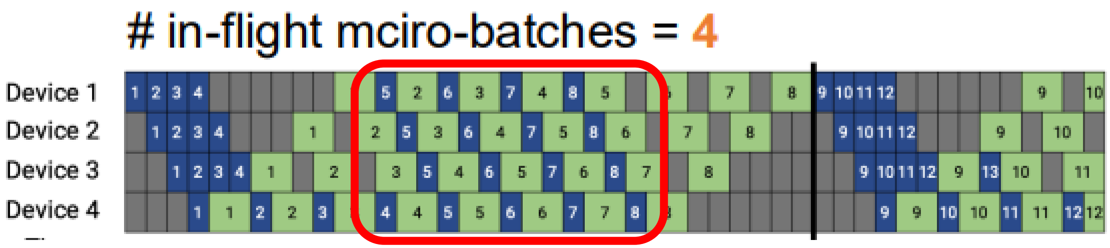
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

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



Pipeline parallelism with GPipe's schedule



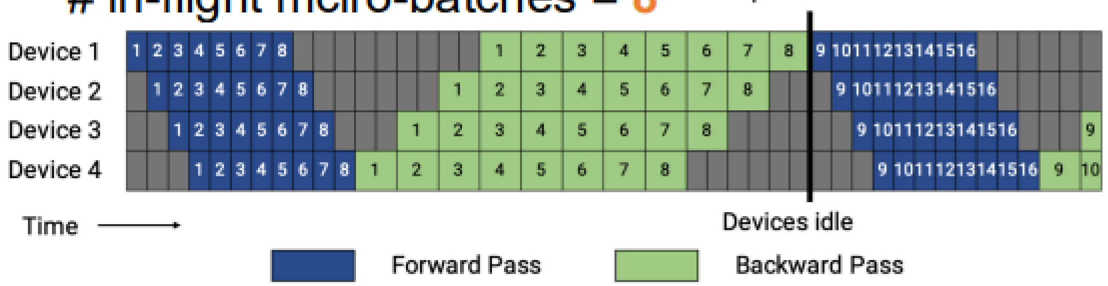
Pipeline parallelism with 1F1B schedule

PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

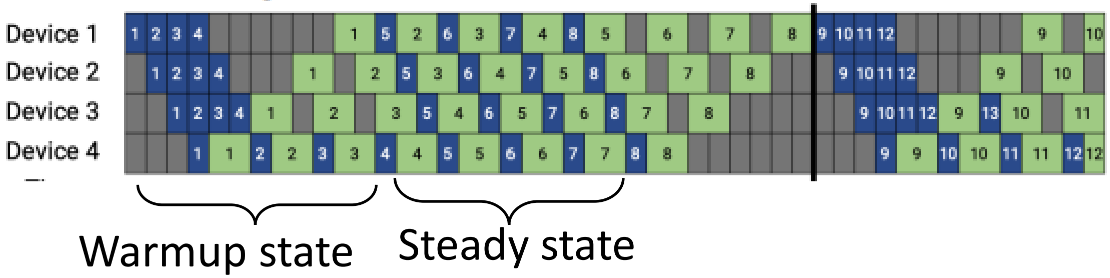
- One-Forward-One-Backward in the steady state

in-flight mciro-batches = 8



Pipeline parallelism with GPipe's schedule

in-flight mciro-batches = 4



Pipeline parallelism with 1F1B schedule

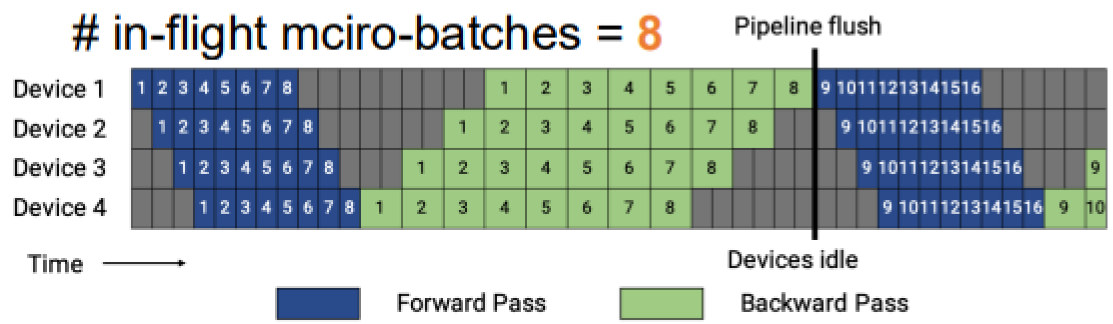
PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Pipeline Parallelism with 1F1B Schedule

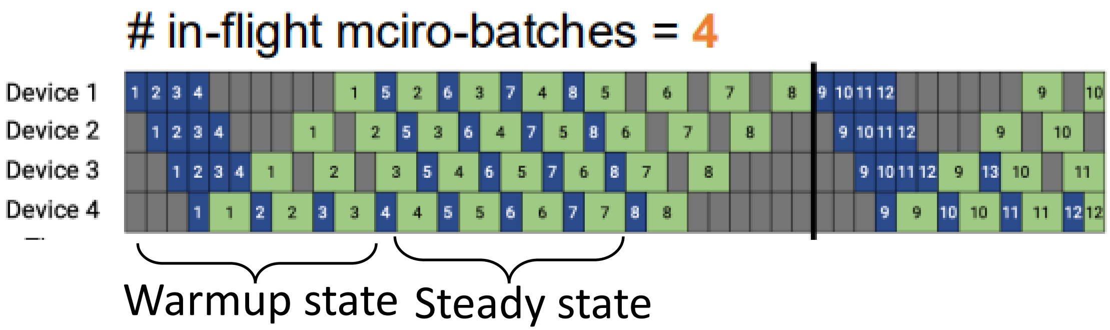
- One-Forward-One-Backward in the steady state

- Reduce memory footprint of pipeline parallelism
- Doesn’t reduce pipeline bubble

Can we reduce pipeline bubble?



Pipeline parallelism with GPipe’s schedule

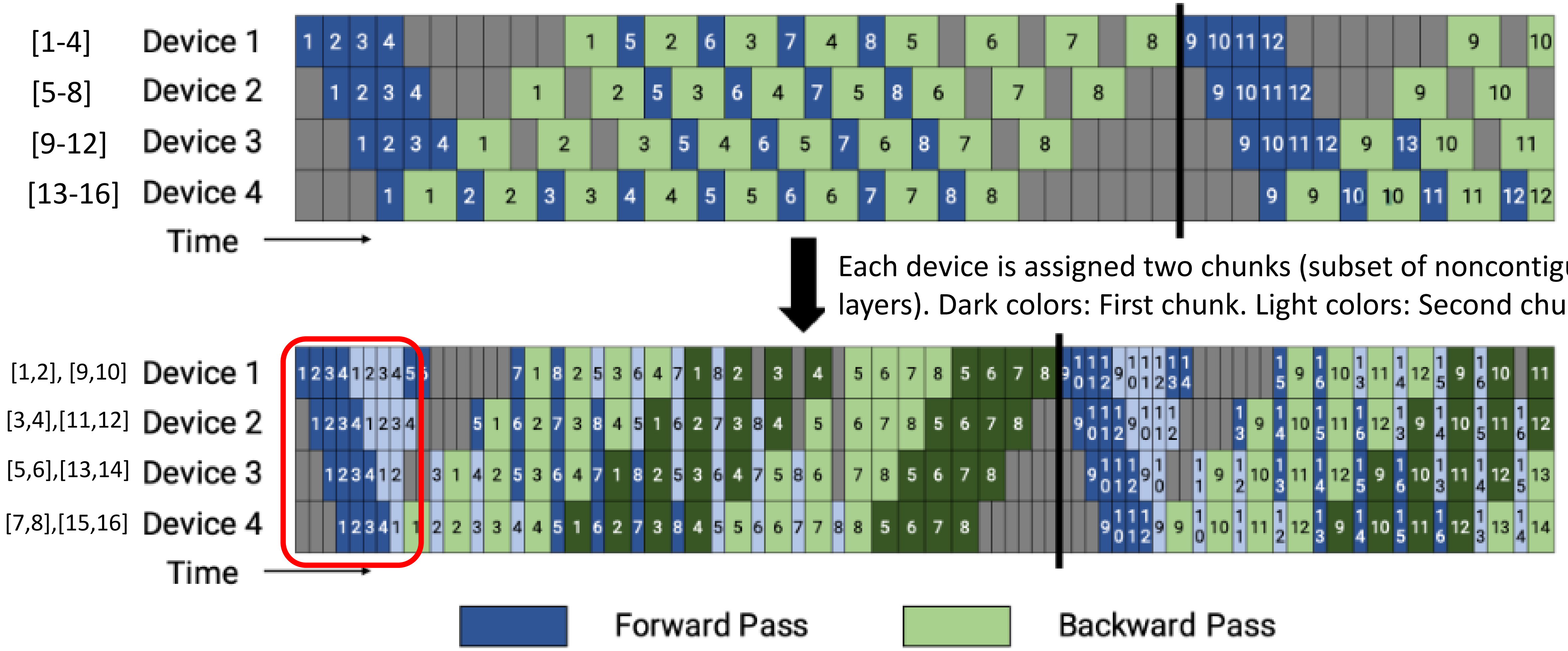


Pipeline parallelism with 1F1B schedule

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

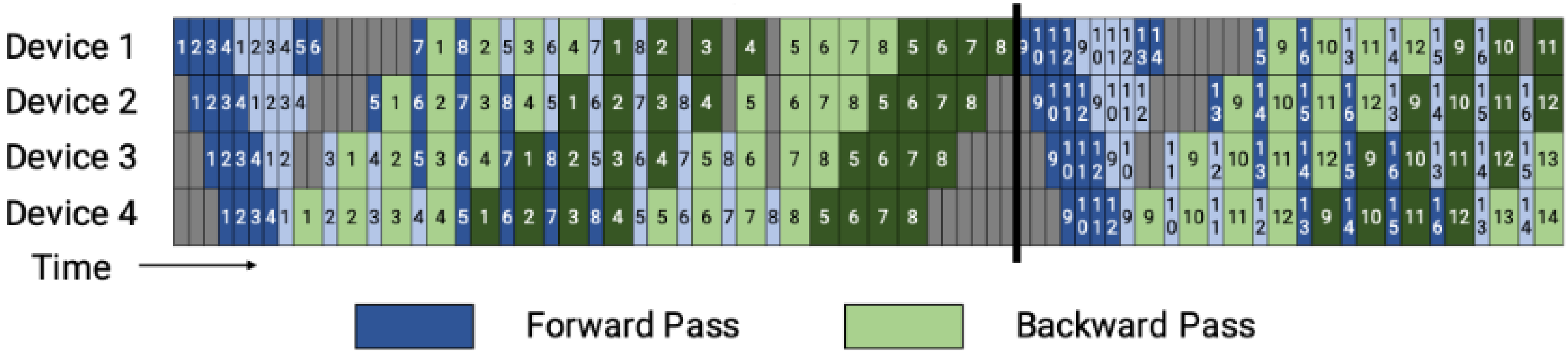


Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM (SC'21)

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

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



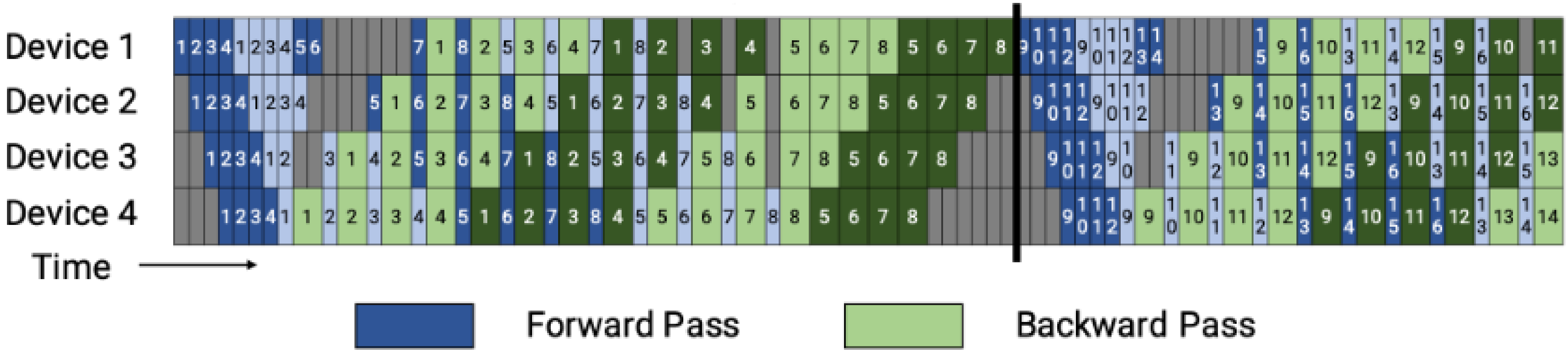
Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM (SC'21)

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?

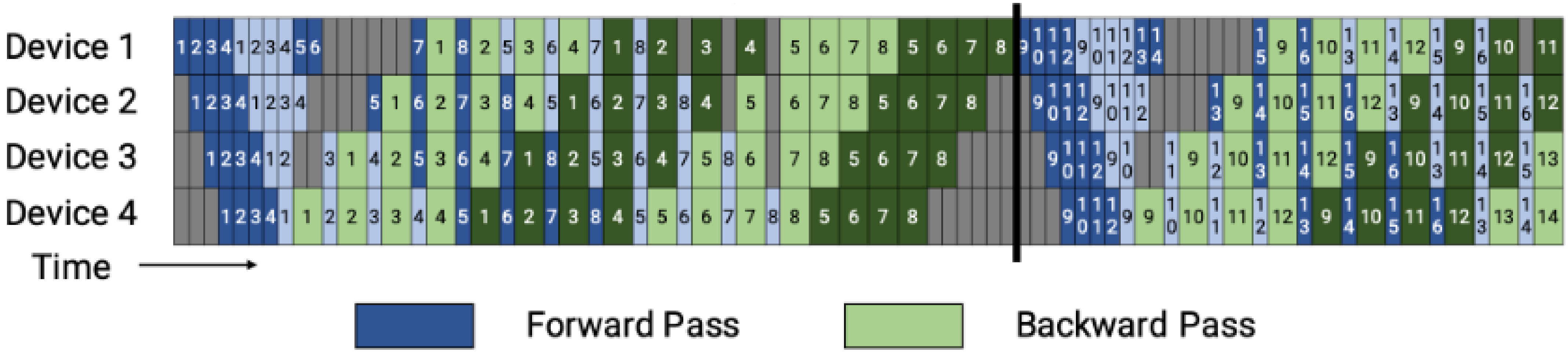


Pipeline Parallelism with Interleaved 1F1B Schedule

- 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



Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM (SC'21)

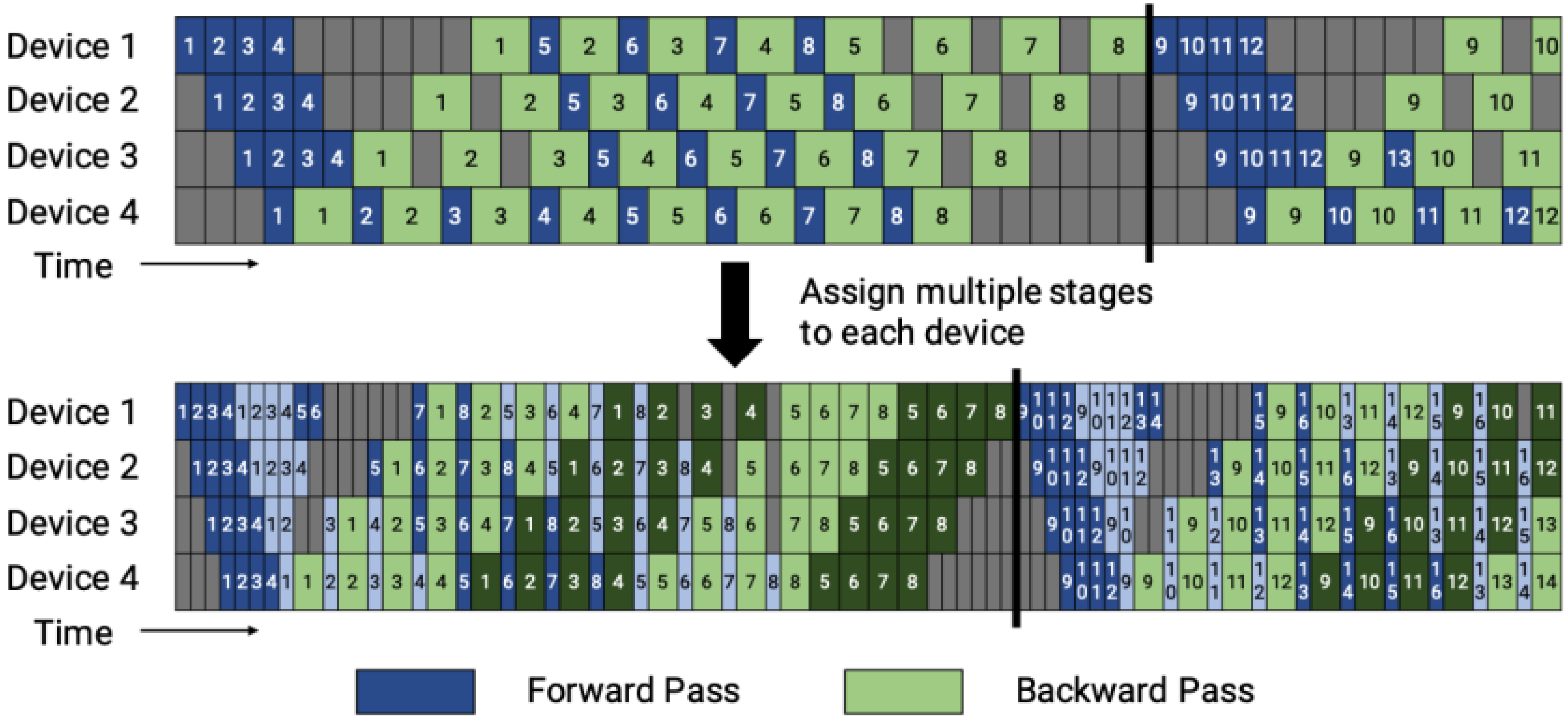
Pipeline Parallelism with Interleaved 1F1B Schedule

Pipeline parallelism with 1F1B Schedule

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

Pipeline parallelism with interleaved 1F1B Schedule

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

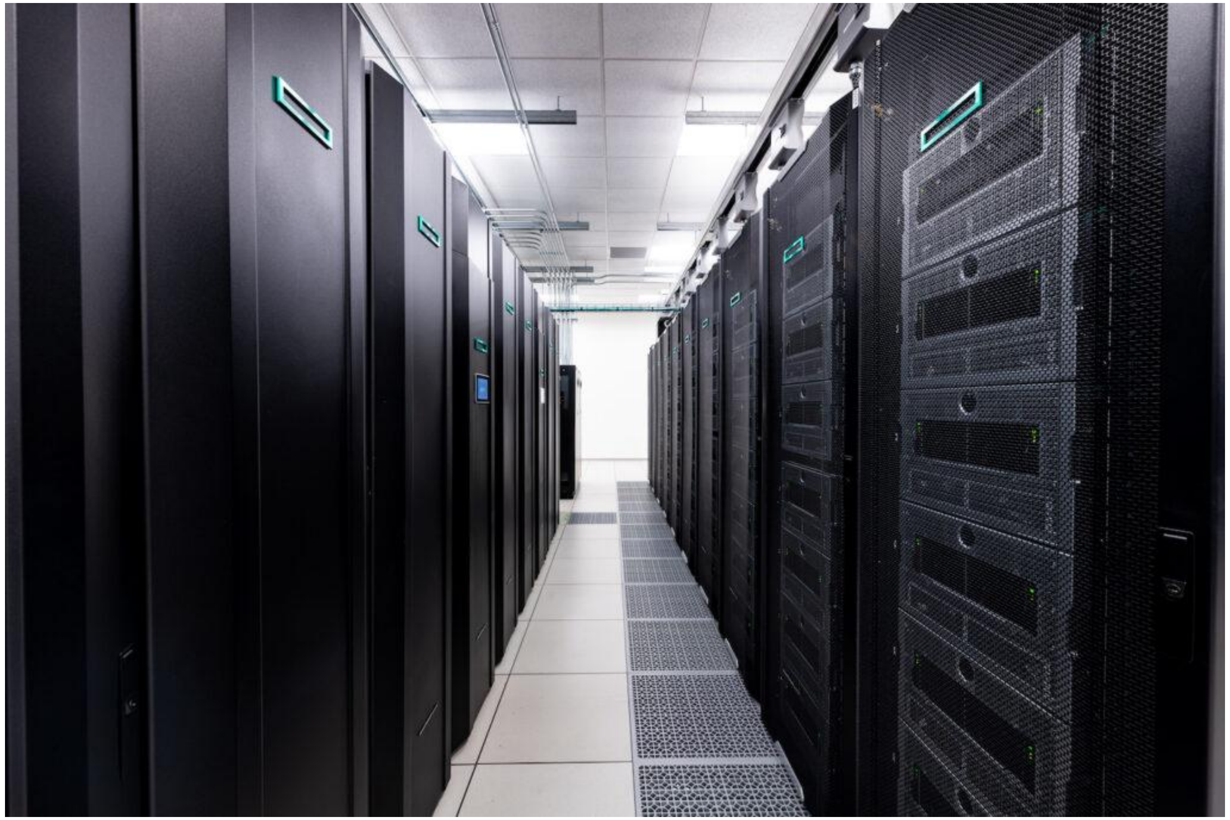


Questions?

Hardware: Delta

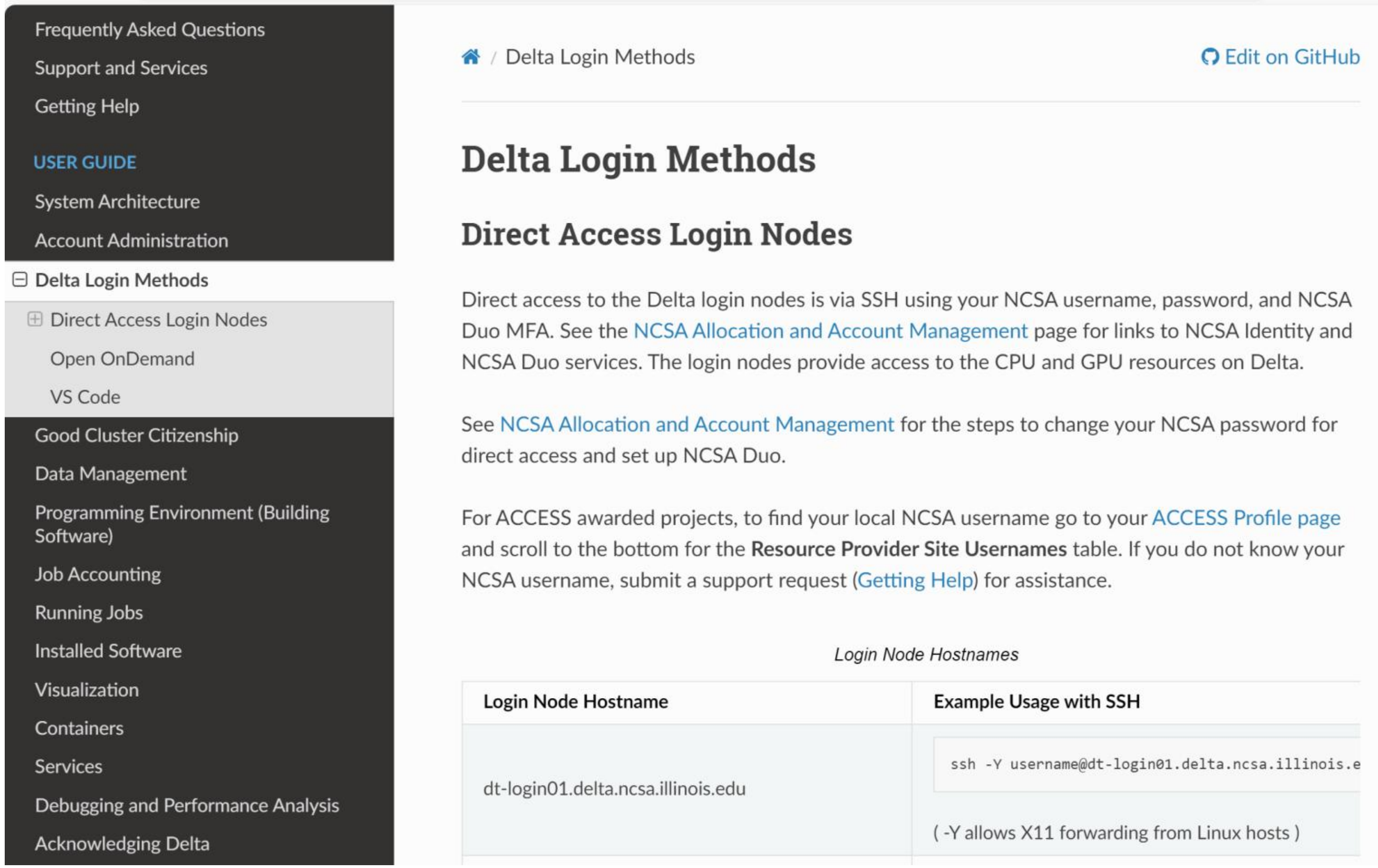
- Home page: https://www.ncsa.illinois.edu/research/project-highlights/delta/

- 100 quad A100 GPU node, each with 4 A100
- 100 quad A40 GPU node, each with 4 A40
- 5 8-way A100 GPU, each with 8 A100
- 1 MI100 node, 8 MI100



Delta Onboarding

- https://docs.ncsa.illinois.edu/systems/delta/en/latest/user_guide/accessing.html



Frequently Asked Questions
Support and Services
Getting Help

USER GUIDE
System Architecture
Account Administration
⊟ Delta Login Methods
⊞ Direct Access Login Nodes
Open OnDemand
VS Code
Good Cluster Citizenship
Data Management
Programming Environment (Building Software)
Job Accounting
Running Jobs
Installed Software
Visualization
Containers
Services
Debugging and Performance Analysis
Acknowledging Delta

Delta Login Methods — Edit on GitHub

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

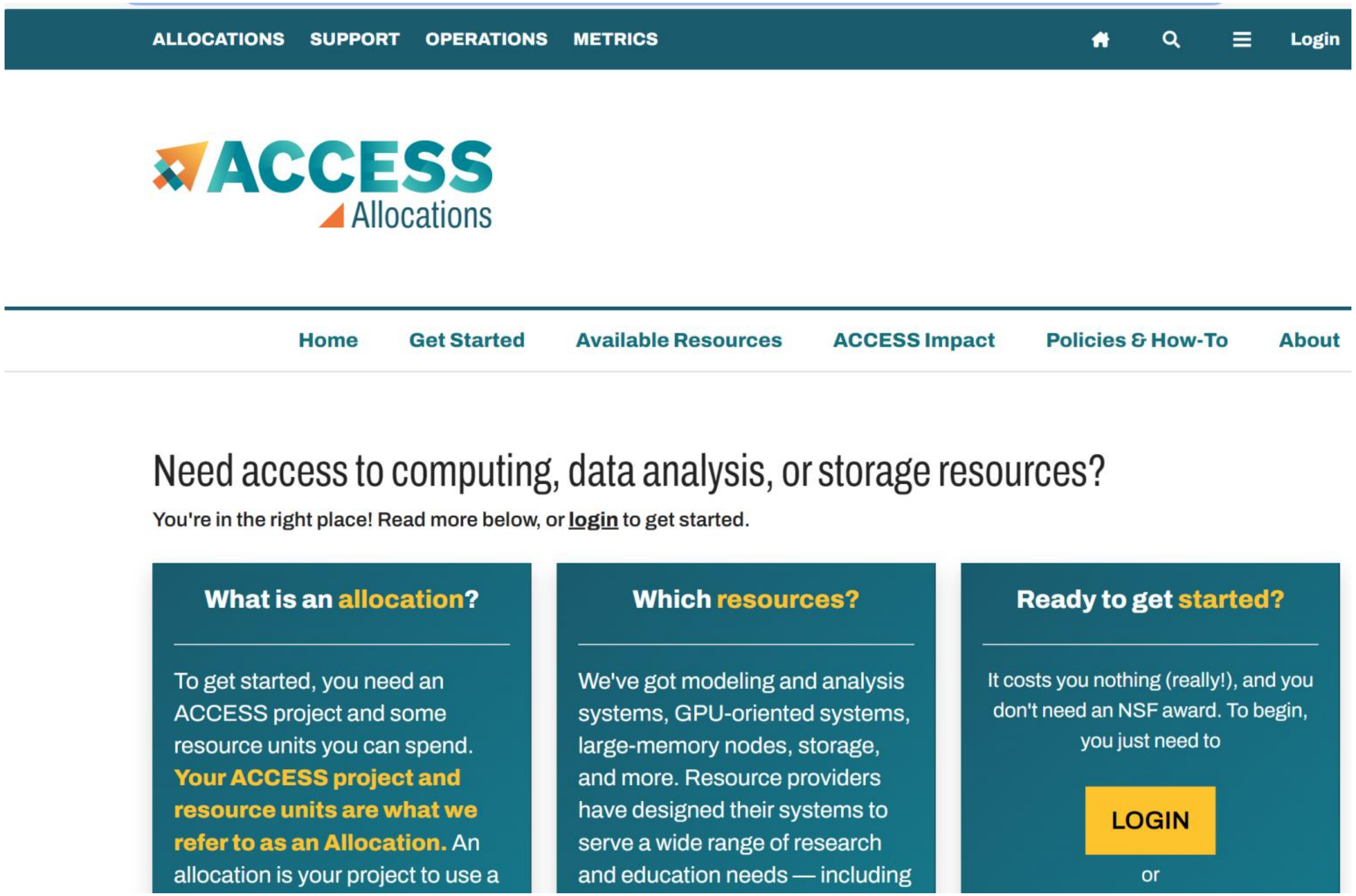
Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)

Step 1: Create ACCESS ID

- Register an ACCESS id at: https://access-ci.org/ (top right-hand corner)

- After you register, send the instructor your ACCESS id. The instructor will add you to access to his GPU allocation.



Step 2: Get Started

- Delta uses Slurm to manage jobs/GPUs
- Please watch this tutorial video: Getting Started on NCSA's Delta Supercomputer.
- After that, you may want to check Delta User Documentation — UIUC NCSA Delta User Guide (illinois.edu).
- Please learn how to use slurm to get GPUs: Slurm Workload Manager - Quick Start User Guide (schedmd.com).

Step 3: SSH Login

- You shall use ssh to login to the node: Delta Login Methods — UIUC NCSA Delta User Guide (illinois.edu).
- For instance, you can use commands such as “srun -A bcjw-delta-gpu --time=00:30:00 --nodes=1 --ntasks-per-node=16 --partition=gpuA100x4,gpuA40x4 --gpus=1 --mem=32g --pty /bin/bash”
- Maintaining Persistent Sessions: tmux

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)
dt-login02.delta.ncsa.illinois.edu	`ssh -l username dt-login02.delta.ncsa.illinois.e` (-l username alt. syntax for `user@host`)
login.delta.ncsa.illinois.edu (round robin DNS name for the set of login nodes)	`ssh username@login.delta.ncsa.illinois.edu`

Additional Info

- It is the instructor's own research allocation, and it has a limit. So please be mindful when using GPU resources.
  - Avoid allocating too many GPUs at once
  - Turn off the job when you are not using the GPUs

- The allocation has 500 GB of storage in total (shared by the class and other students in the instructor's lab)
  - Please avoid downloading large data files and super large model checkpoints, e.g., one llama7b checkpoint consumes roughly 14GB.

Course Project (Reproducibility Challenge)

4-credit undergraduate students, 3-credit graduate students
Some suggestions below, but it can be any machine learning system related papers that you are interested in

GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings	https://github.com/zilliztech/GPTCache
RouteLLM: Learning to Route LLMs with Preference Data	https://github.com/lm-sys/RouteLLM
LLM-QAT: Data-Free Quantization Aware Training for Large Language Models	https://github.com/facebookresearch/LLM-QAT
Speculative decoding in vLLM	https://docs.vllm.ai/en/v0.5.5/models/spec_decode.html
REST: Retrieval-Based Speculative Decoding	https://github.com/FasterDecoding/REST
MemGPT: Towards LLMs as Operating Systems	https://github.com/letta-ai/letta
…	…

Course Project (Open-End Research Project)

4-credit graduate students

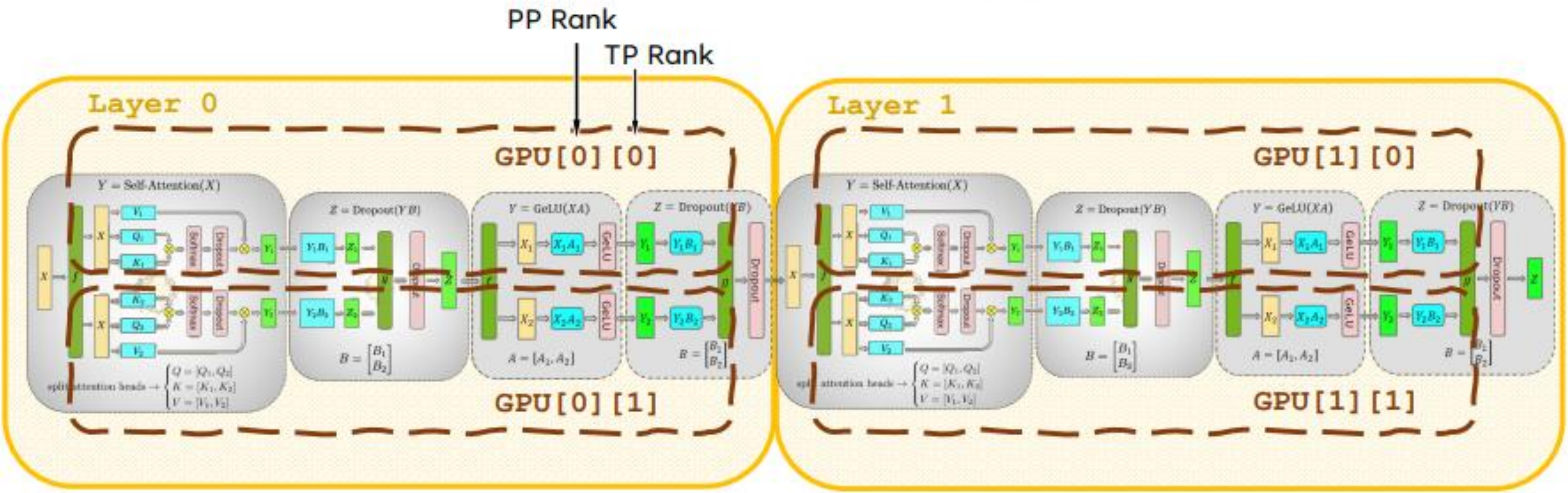
- Benchmark and analyze important DL workloads to understand their performance gap and identify important angles to optimize their performance.
- Apply and evaluate how existing solutions work in the context of emerging AI/DL workloads.
- Design and implement new algorithms that are both theoretically and practically efficient.
- Design and implement system optimizations, e.g., parallelism, cache-locality, IO-efficiency, to improve the compute/memory/communication efficiency of AI/DL workloads.
- Offer customized optimization for critical DL workloads where latency is extremely tight.
- Build library/tool/framework to improve the efficiency of a class of problems.
- Integrate important optimizations into existing frameworks (e.g., DeepSpeed), providing fast and agile inference.
- Combine system optimization with modeling optimizations.
- Combine and leverage hardware resources (e.g., GPU/CPU, on-device memory/DRAM/NVMe/SSD) in a principled way.

Today

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 ⇩

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

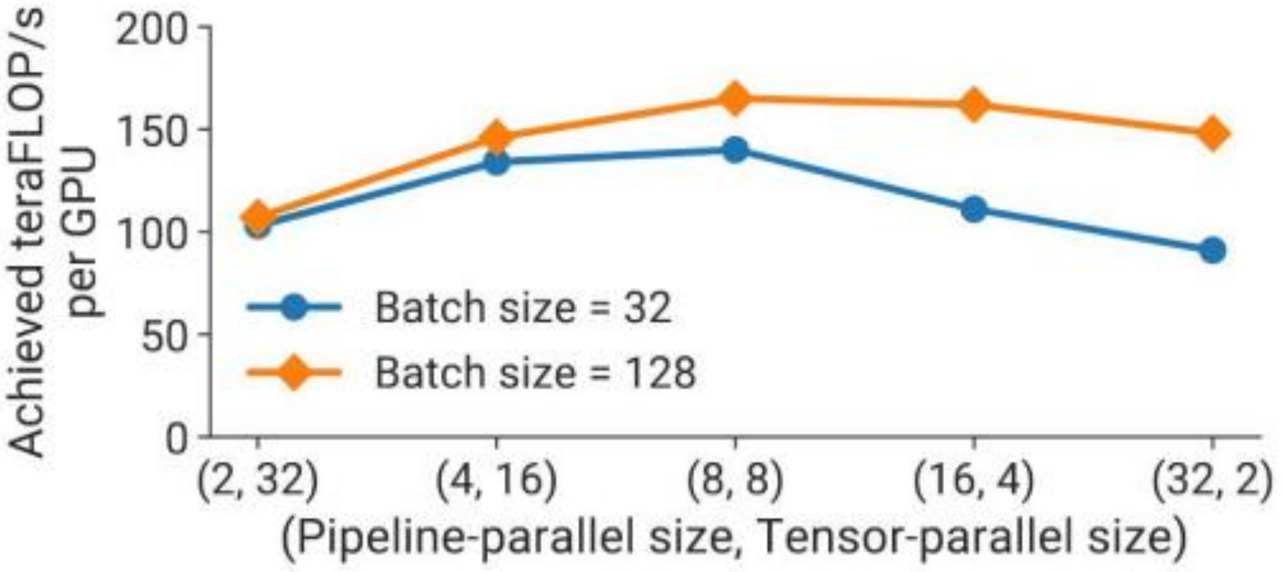
- Communication overhead
  - All-reduce communication for tensor model parallelism is expensive!
  - Especially when cross servers

- (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.

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

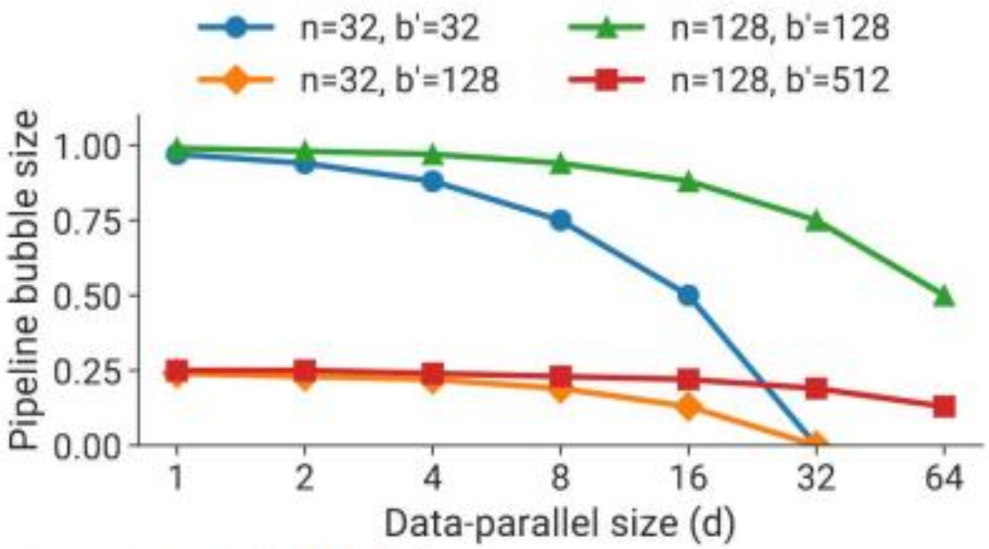


Performance Analysis of Combined Parallelism

- 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*

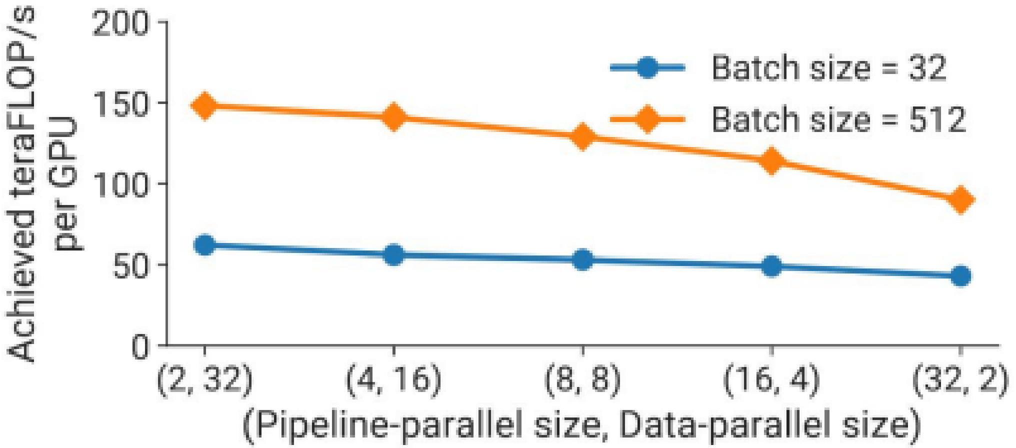


- Data versus Tensor Parallelism
  - DP is less communication heavy than TP
    - All-reduce once per batch vs. All-reduce once per microbatch
  - Tensor parallelism can lead to hardware underutilization

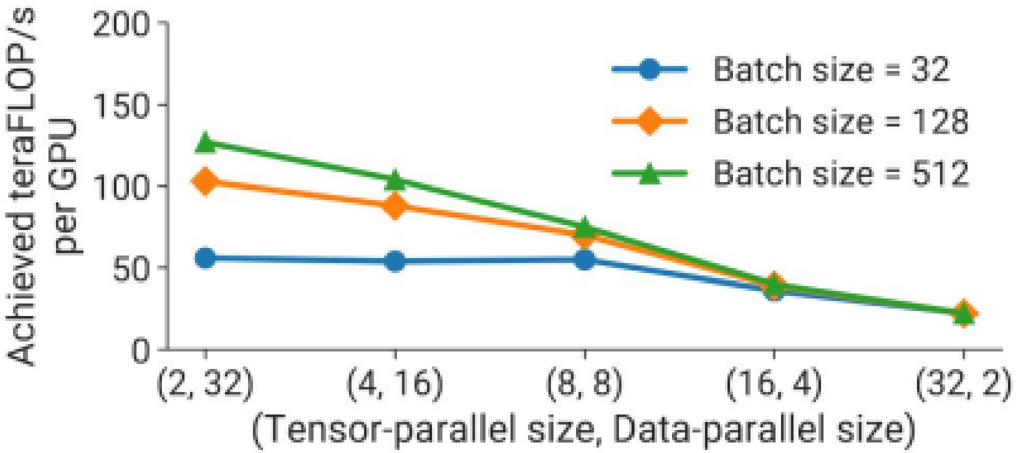
Takeaway #2: Decide tensor-parallel size and pipeline-parallel size based on the GPU memory size; data parallelism can be used to scale to more GPUs.

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

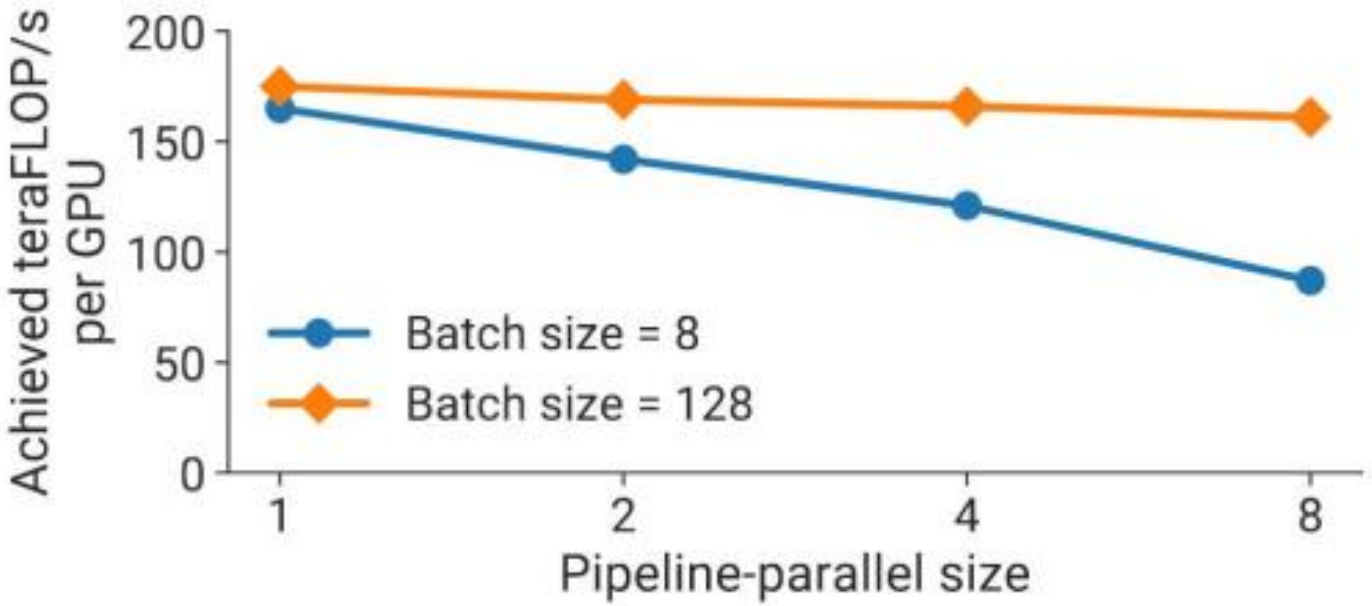


Limitations of data-parallelism:

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

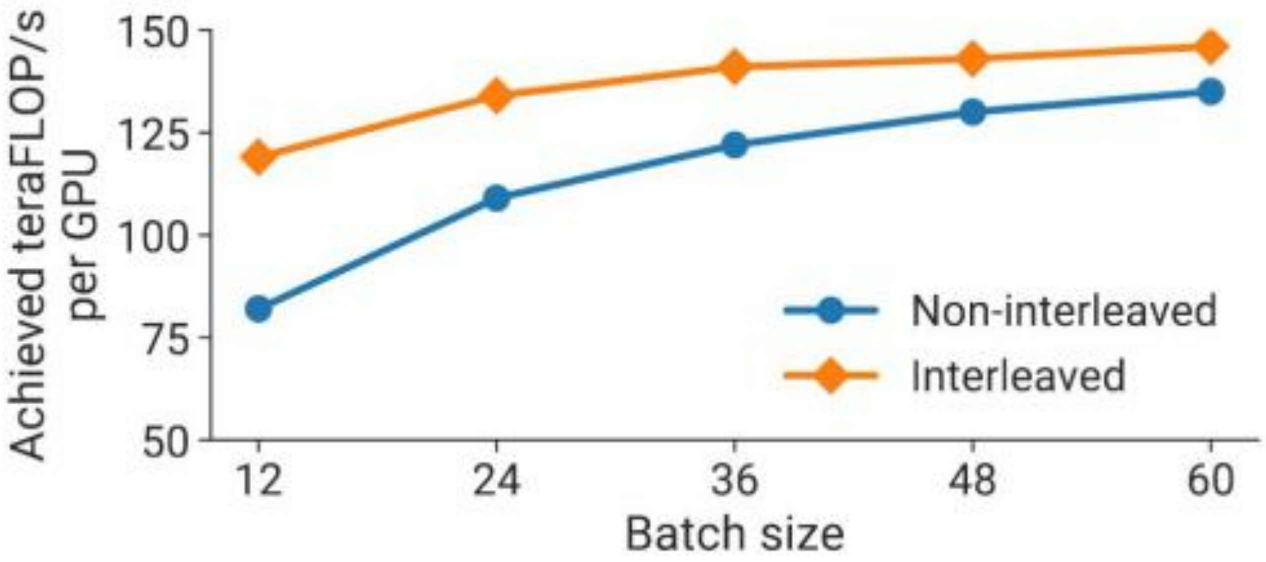
Evaluation - Pipeline Parallelism

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



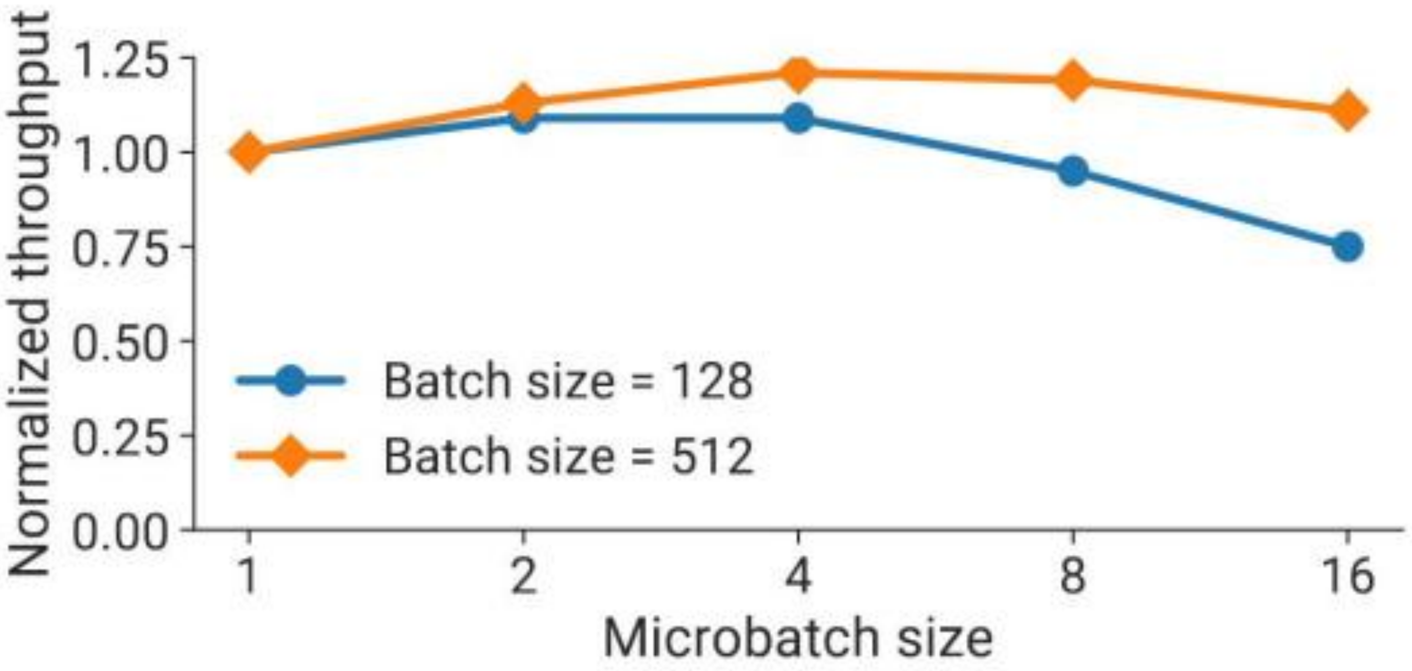
Evaluation - Pipeline Parallelism

- 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



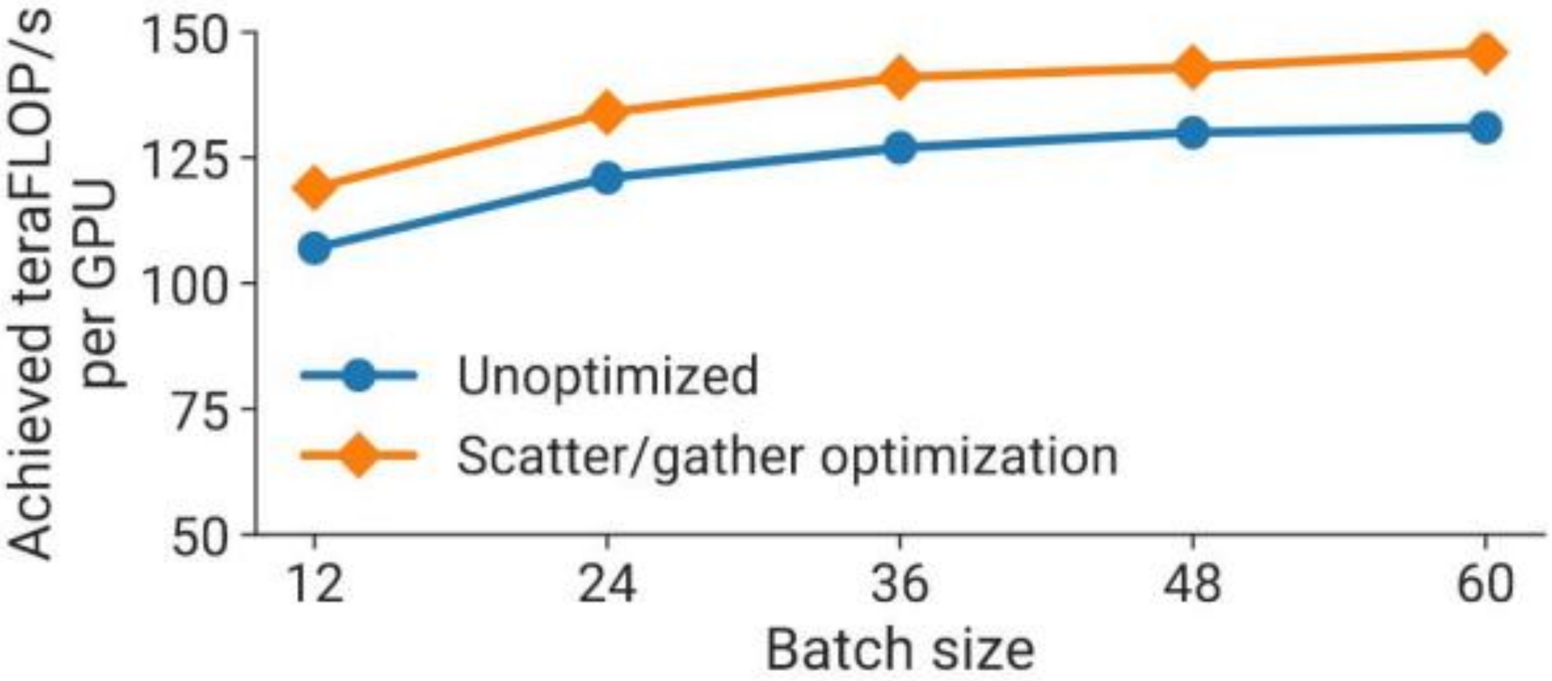
Evaluation - Selection of Microbatch size

- 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

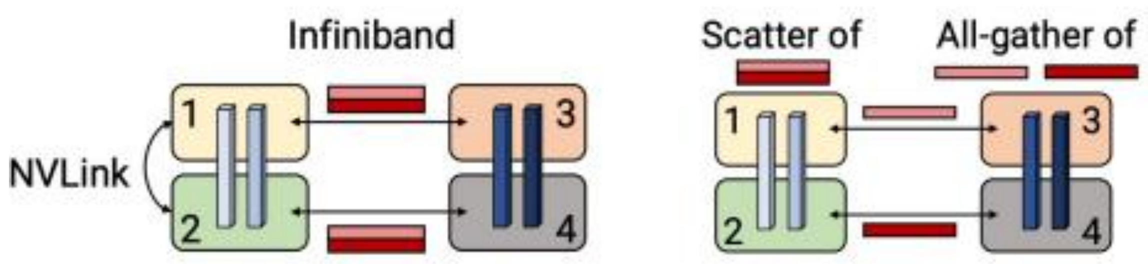
$$\text{End-to-end training time} \approx \frac{8TP}{nX}$$

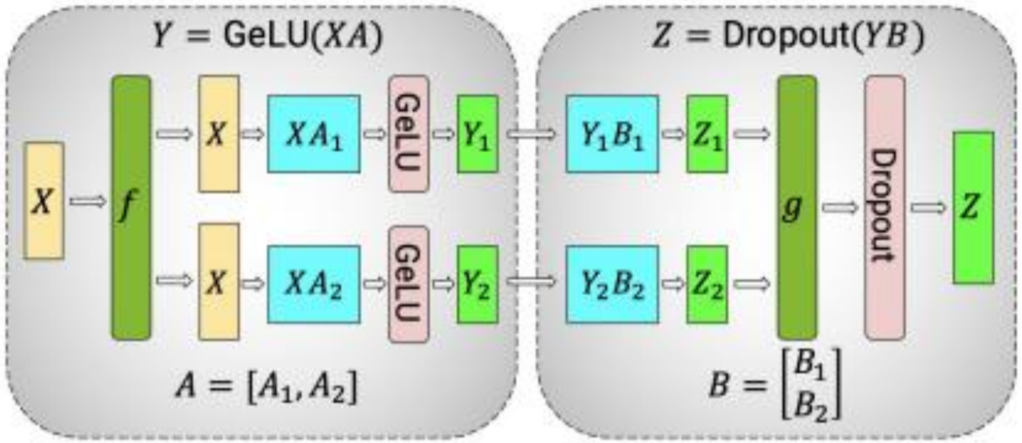
T (billion)	P (billion)	n	X (teraFLOPs/s per GPU)	#Days
300	175	1024	140	34
1000	450	3072	163	84

288 years with a single V100 NVIDIA GPU

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.