Doing more with less: Training large DNN models on commodity servers for the masses

Youjie Li
UIUC

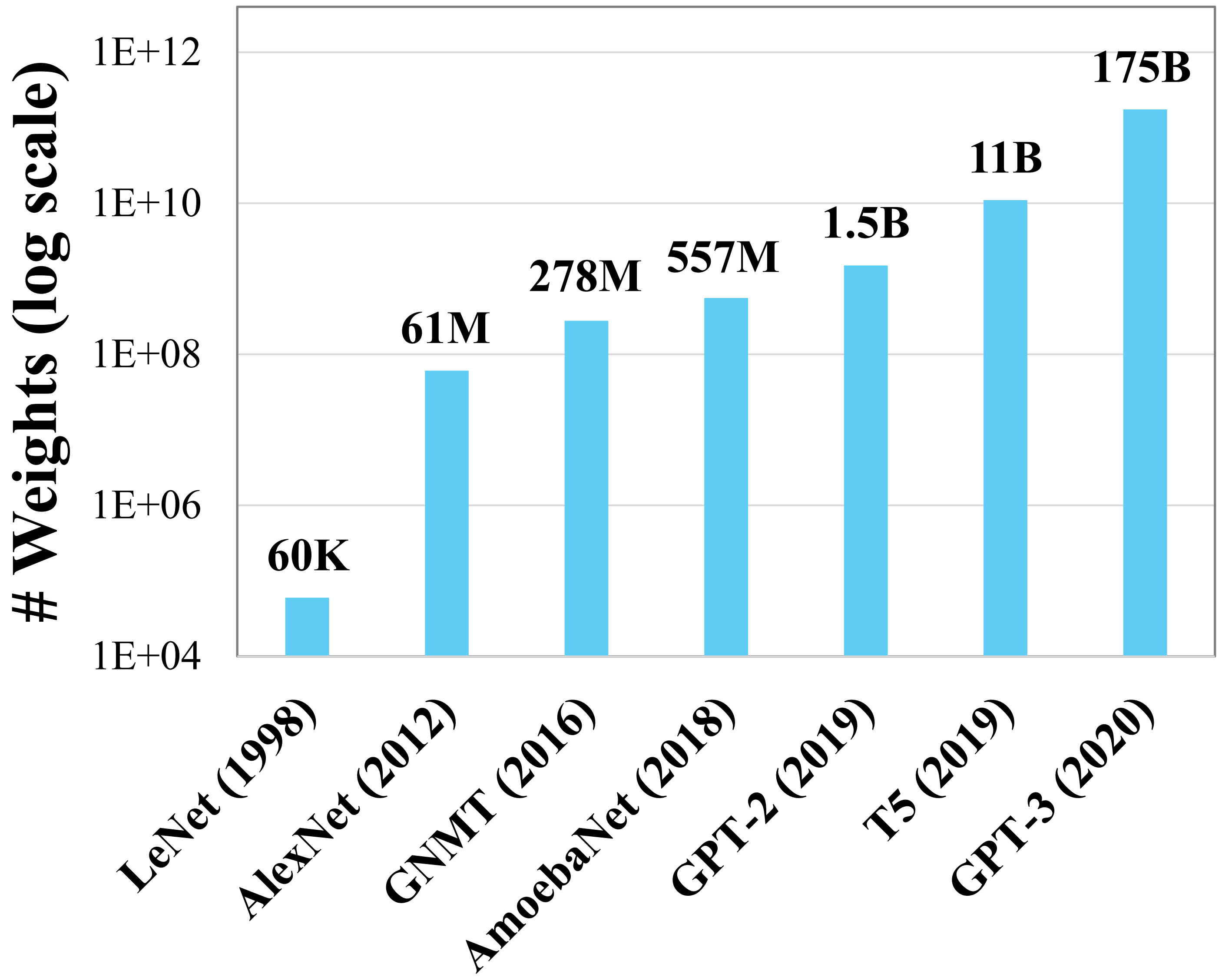
Amar Phanishayee
Microsoft Research

Derek Murray
Microsoft

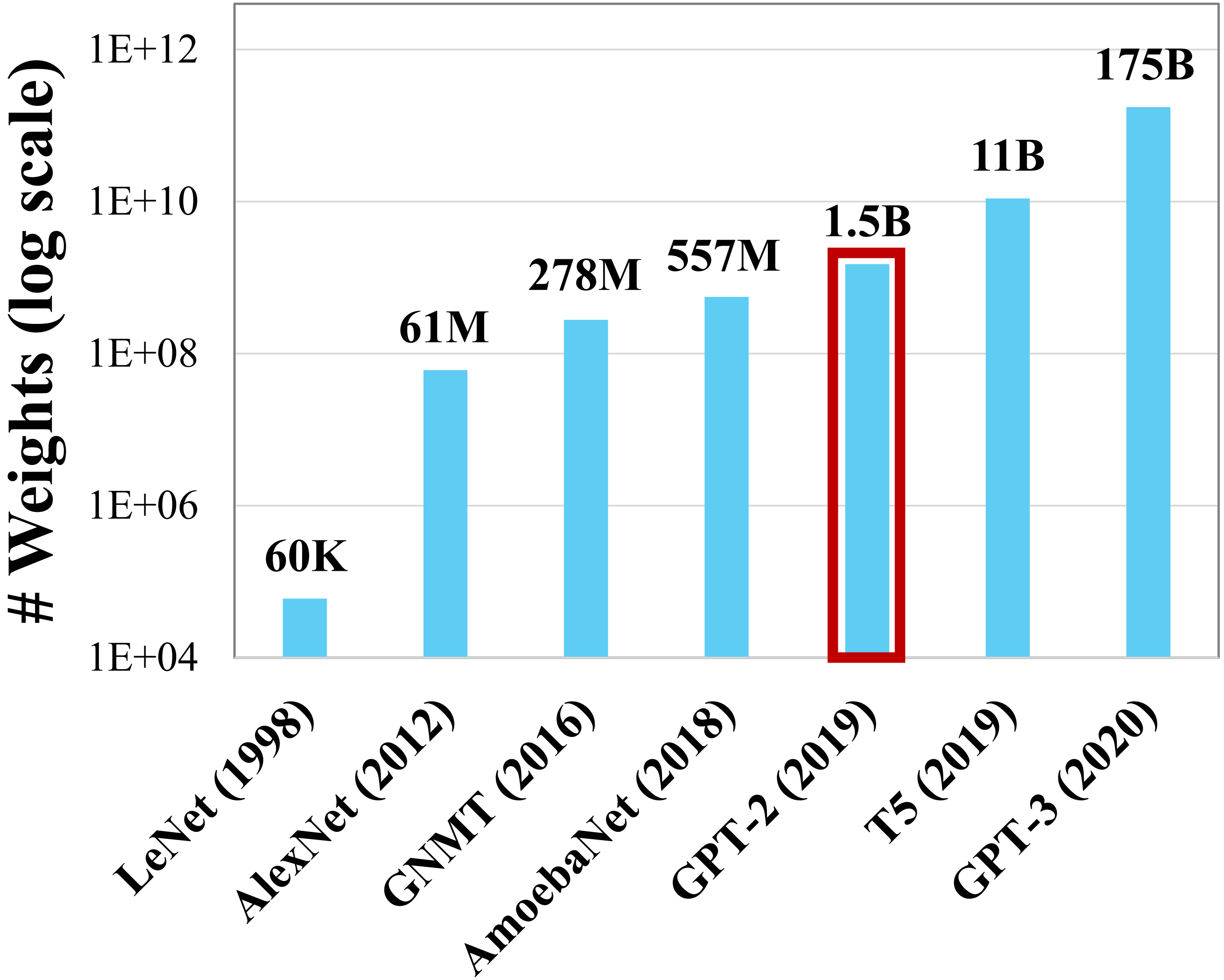
Nam Sung Kim
UIUC

HotOS 2021

DNN model size are growing exponentially!



DNN model size are growing exponentially!

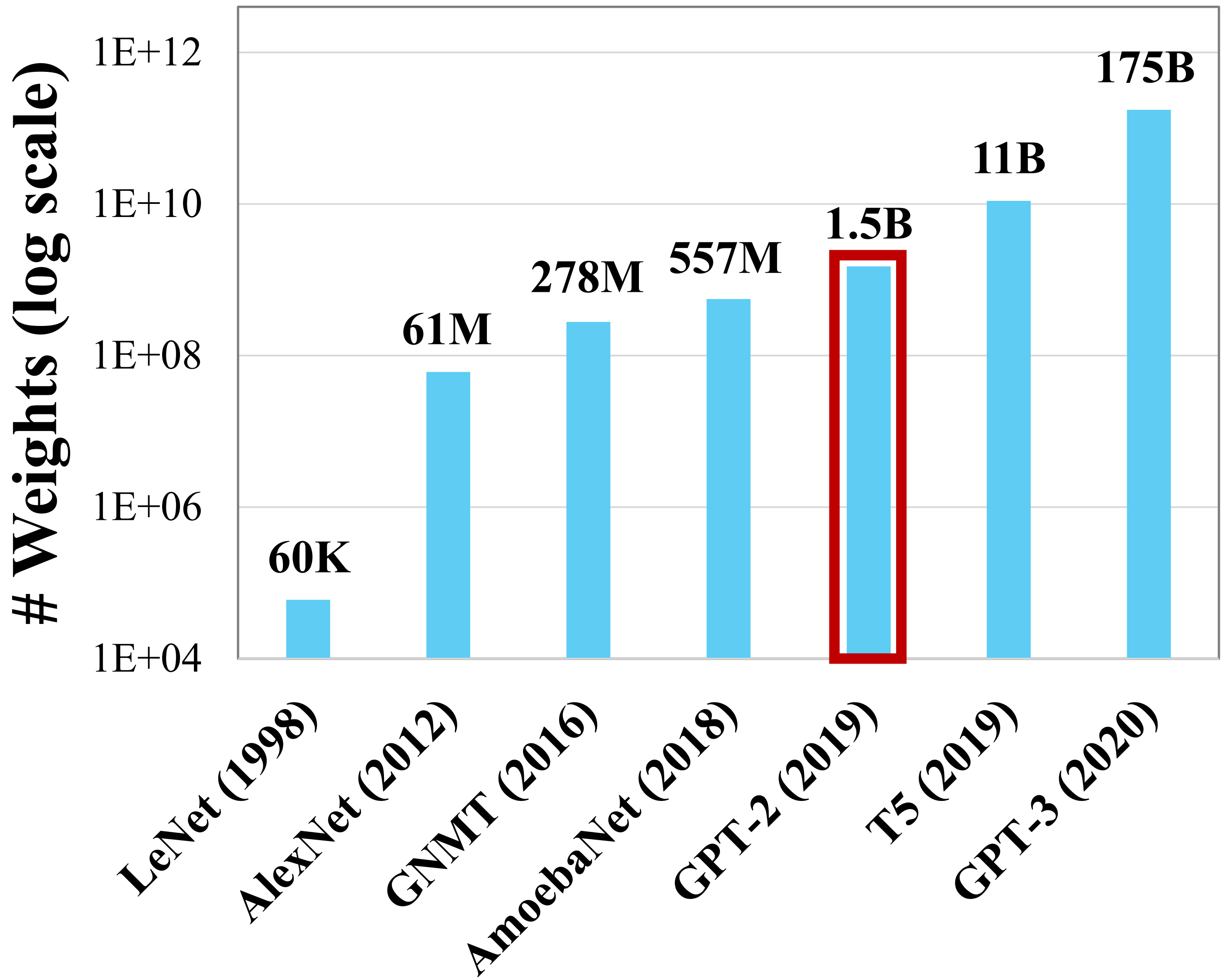


128 GB footprint for training GPT2*

Breakdown	Mem Usage	Percent
Weight	3 GB	2.3%
Gradient	3 GB	2.3%
Optimizer	18 GB	14.1%
Stashing	60 GB	46.9%
Buffer	6 GB	4.7%
Other	38 GB	29.7%
Total	128 GB	100%

*[GPT2, arXiv'19][Zero, SC'20]

DNN model size are growing exponentially!



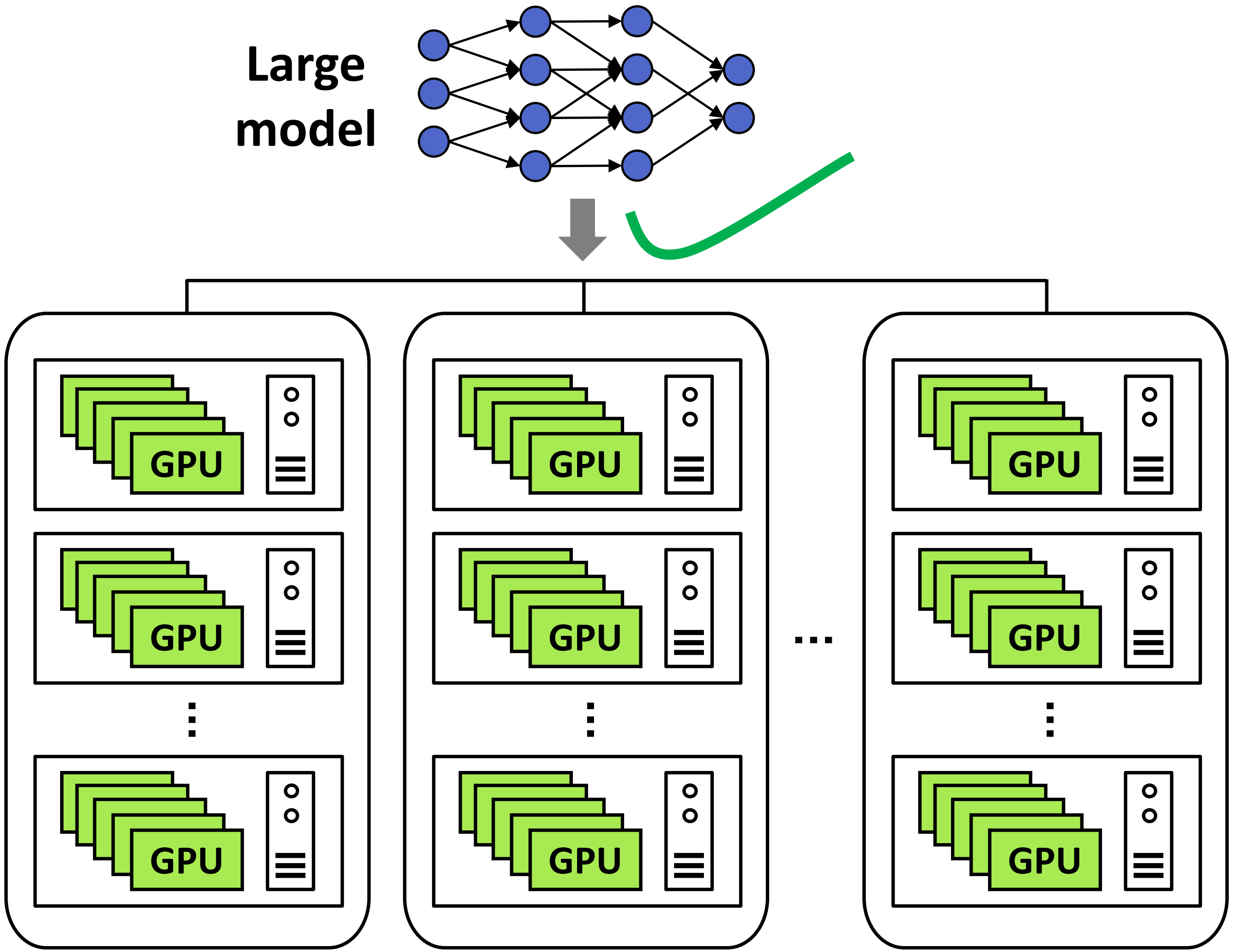
128 GB footprint for training GPT2*

Breakdown	Mem Usage	Percent
Weight	3 GB	2.3%
Gradient	3 GB	2.3%
Optimizer	18 GB	14.1%
Stashing	60 GB	46.9%
Buffer	6 GB	4.7%
Other	38 GB	29.7%
Total	128 GB	100%

Weights are only a fraction of total memory usage!

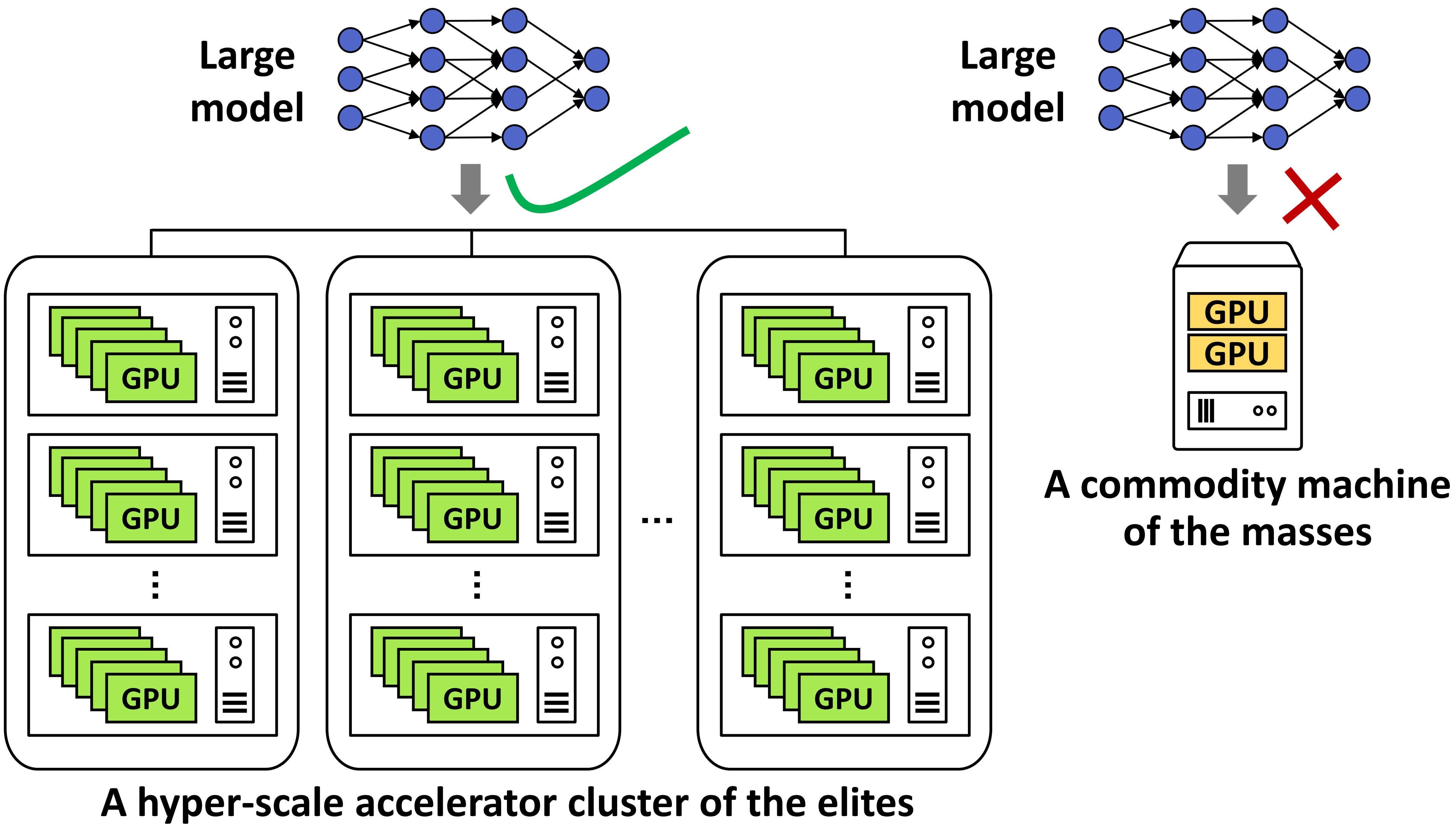
*[GPT2, arXiv'19][Zero, SC'20]

Now, only “elites” can train large models

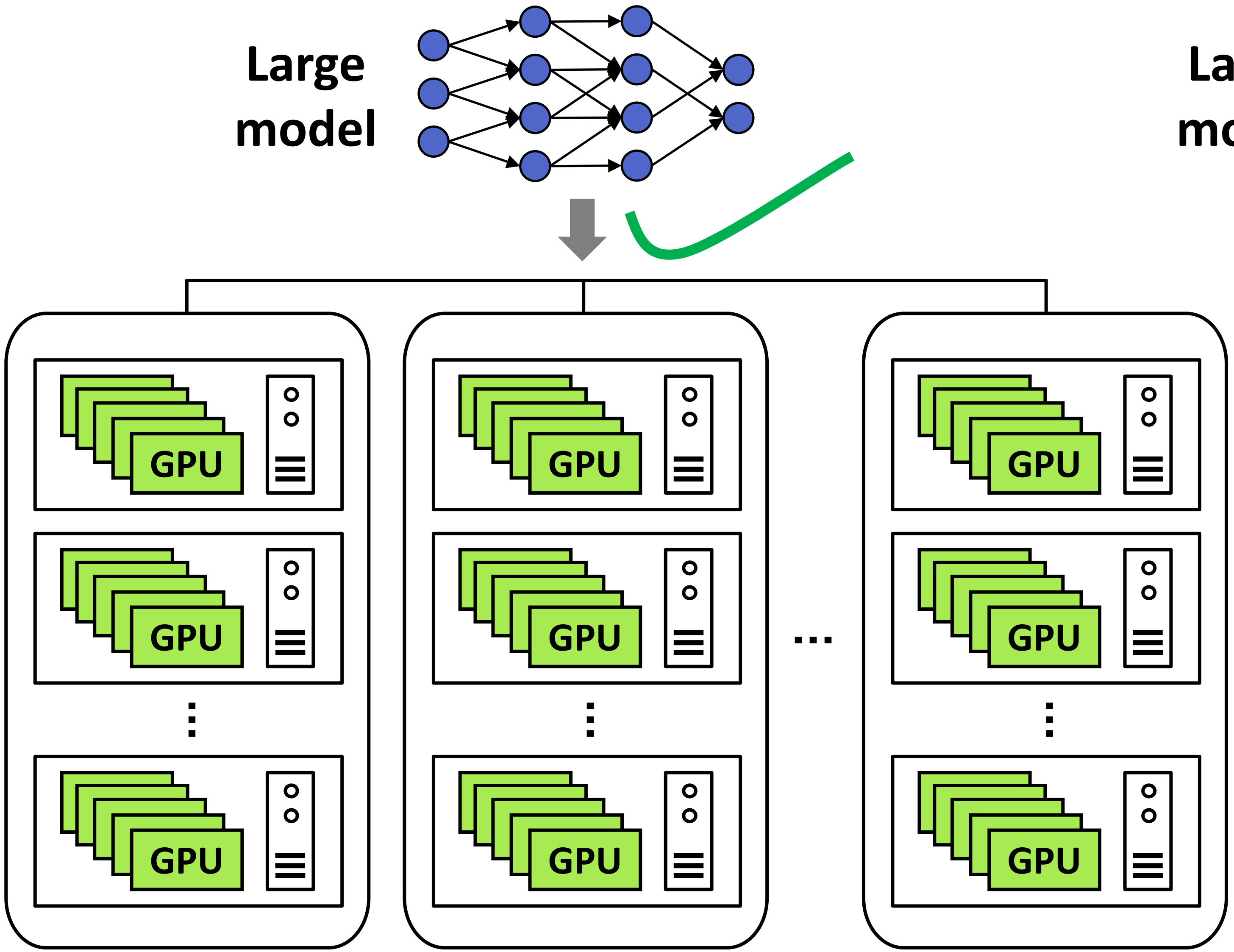


A hyper-scale accelerator cluster of the elites

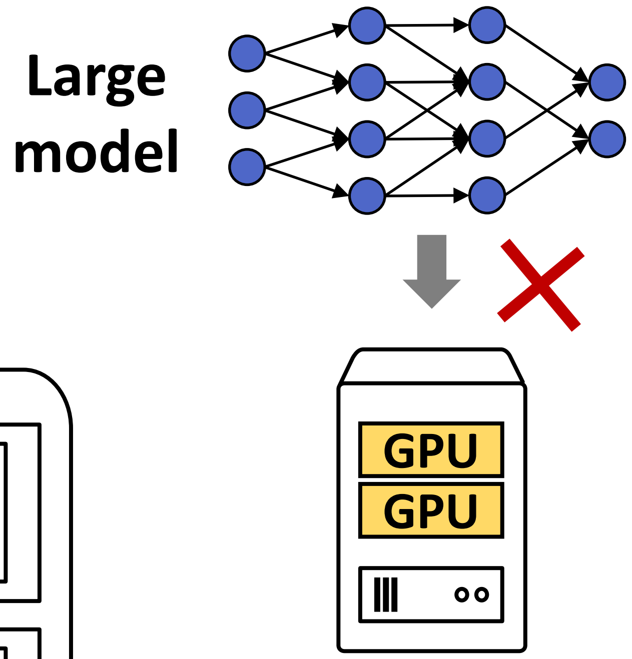
Now, only “elites” can train large models



Now, only "elites" can train large models



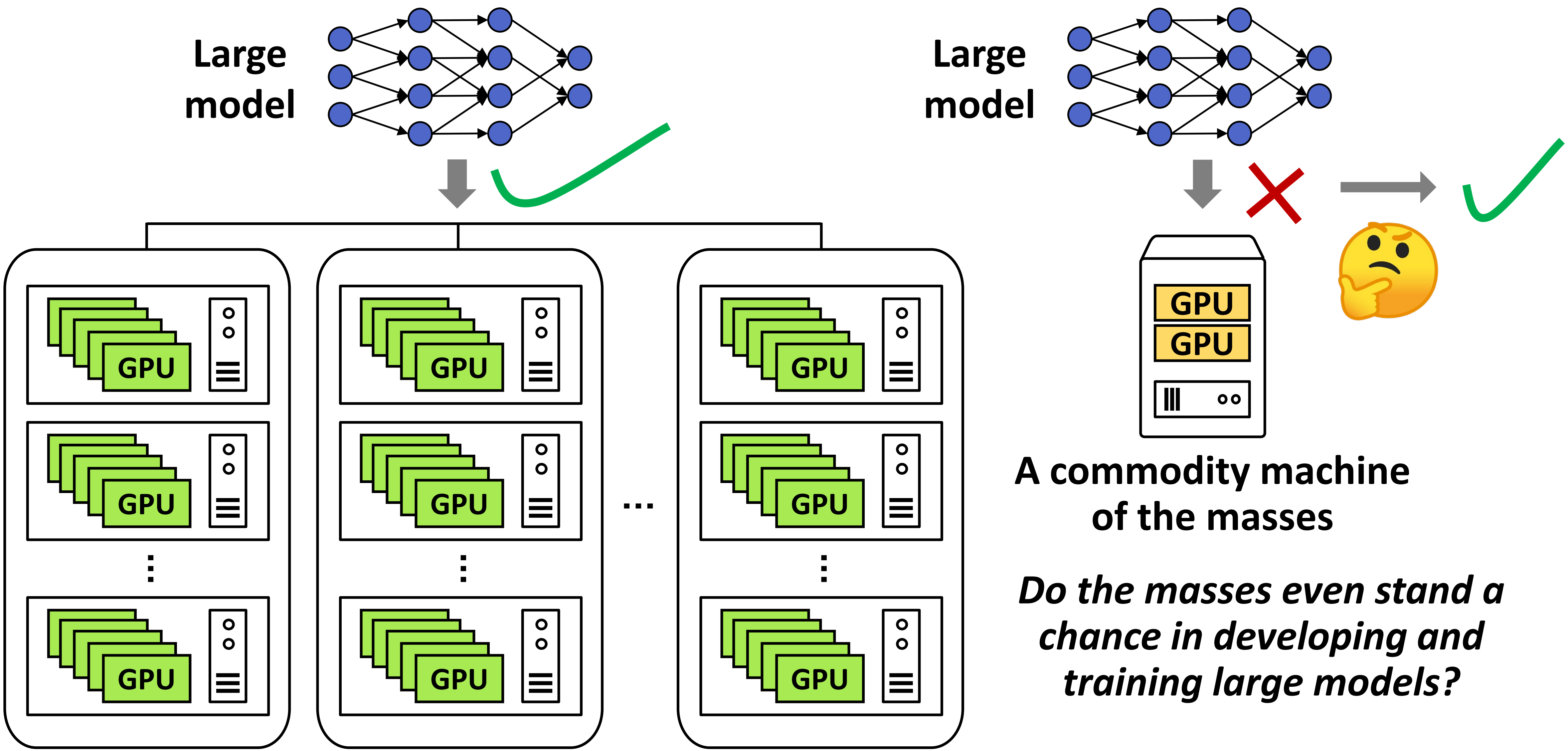
A hyper-scale accelerator cluster of the elites



A commodity machine of the masses

Do the masses even stand a chance in developing and training large models?

Now, only “elites” can train large models



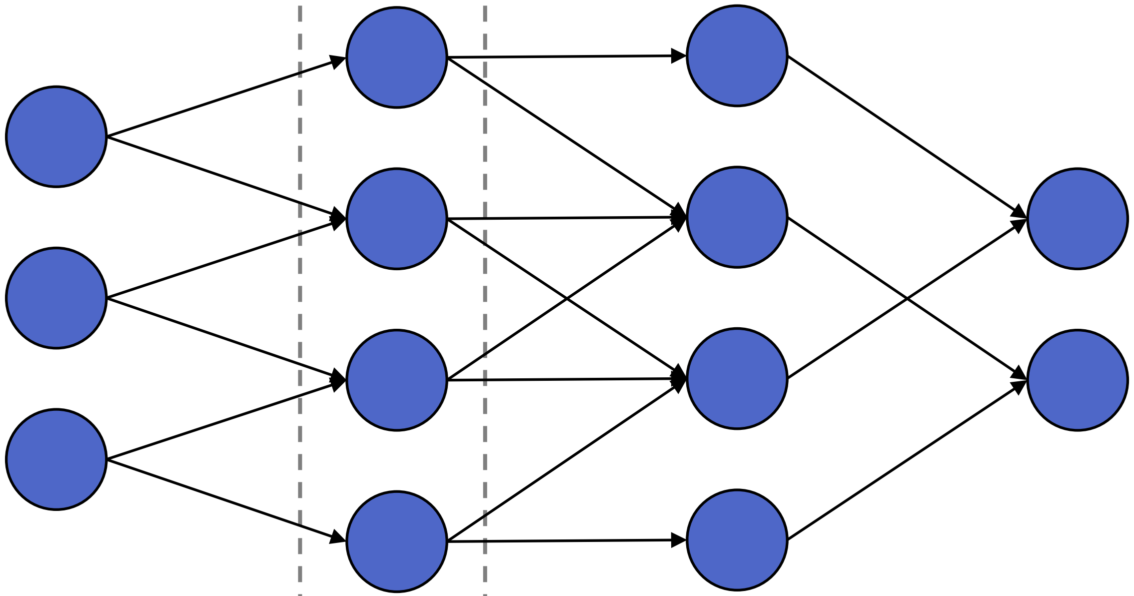
A hyper-scale accelerator cluster of the elites

A commodity machine of the masses

Do the masses even stand a chance in developing and training large models?

The main challenge:

training memory footprint > accelerator memory capacity

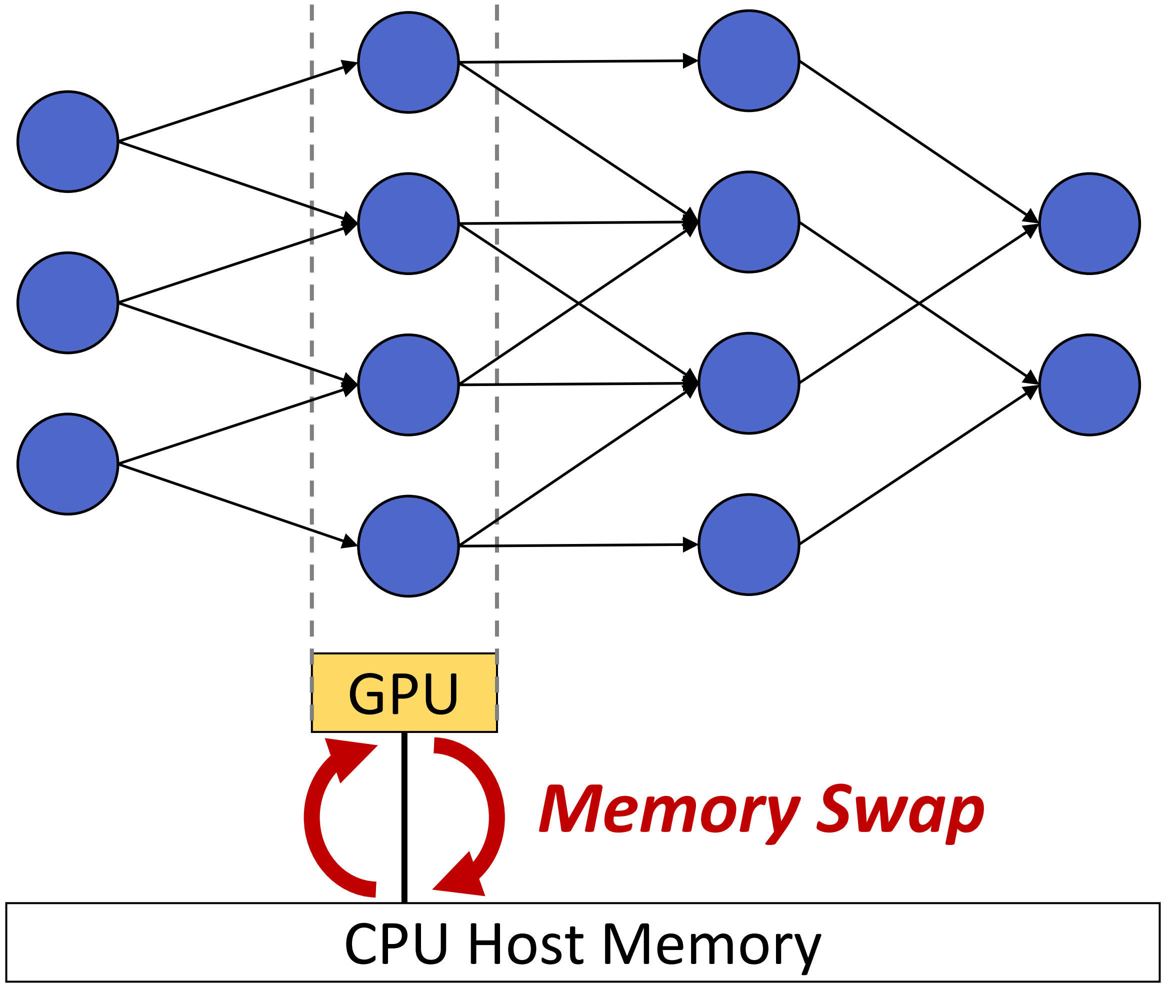


128 GB footprint for training GPT2

GPU

16 GB V100

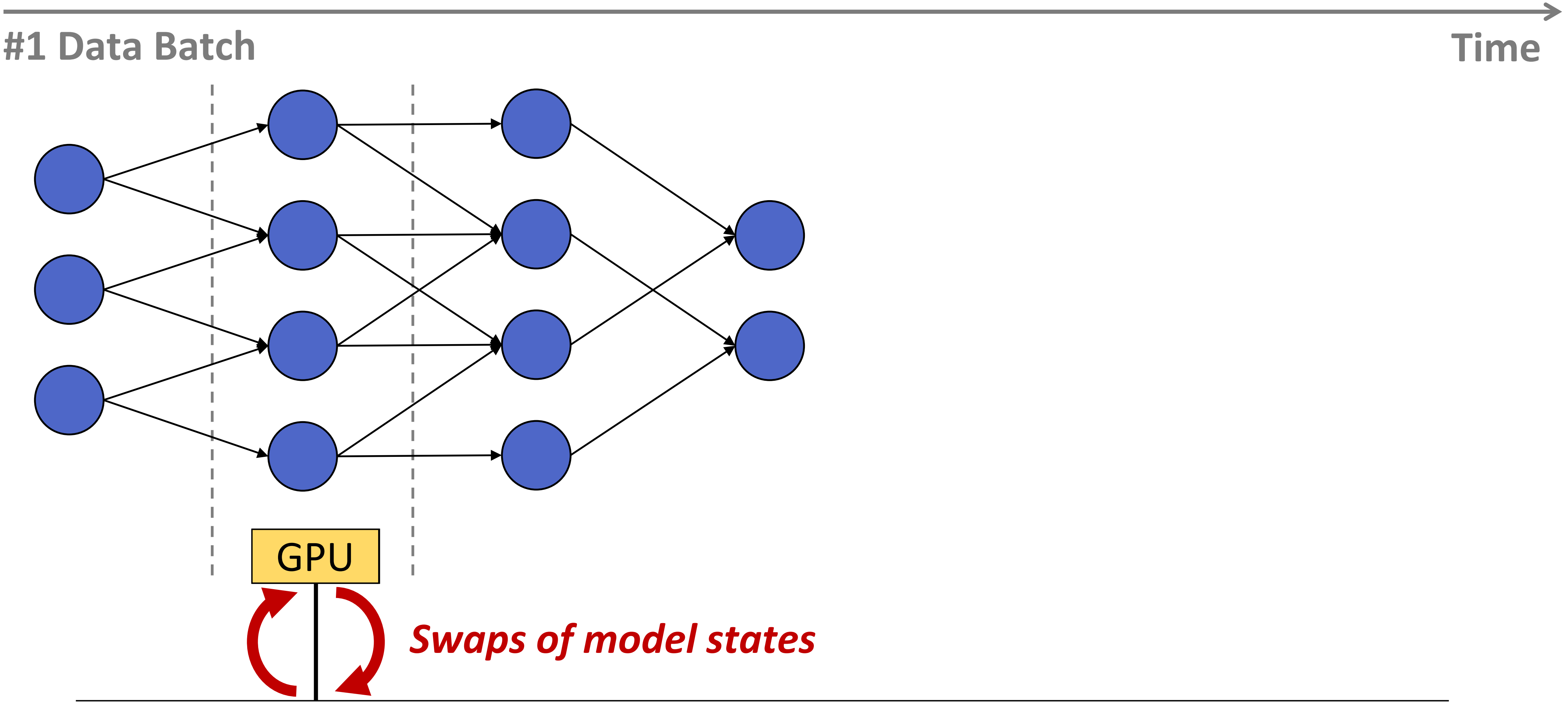
Promising technique #1: *Single-GPU Memory Virtualization**



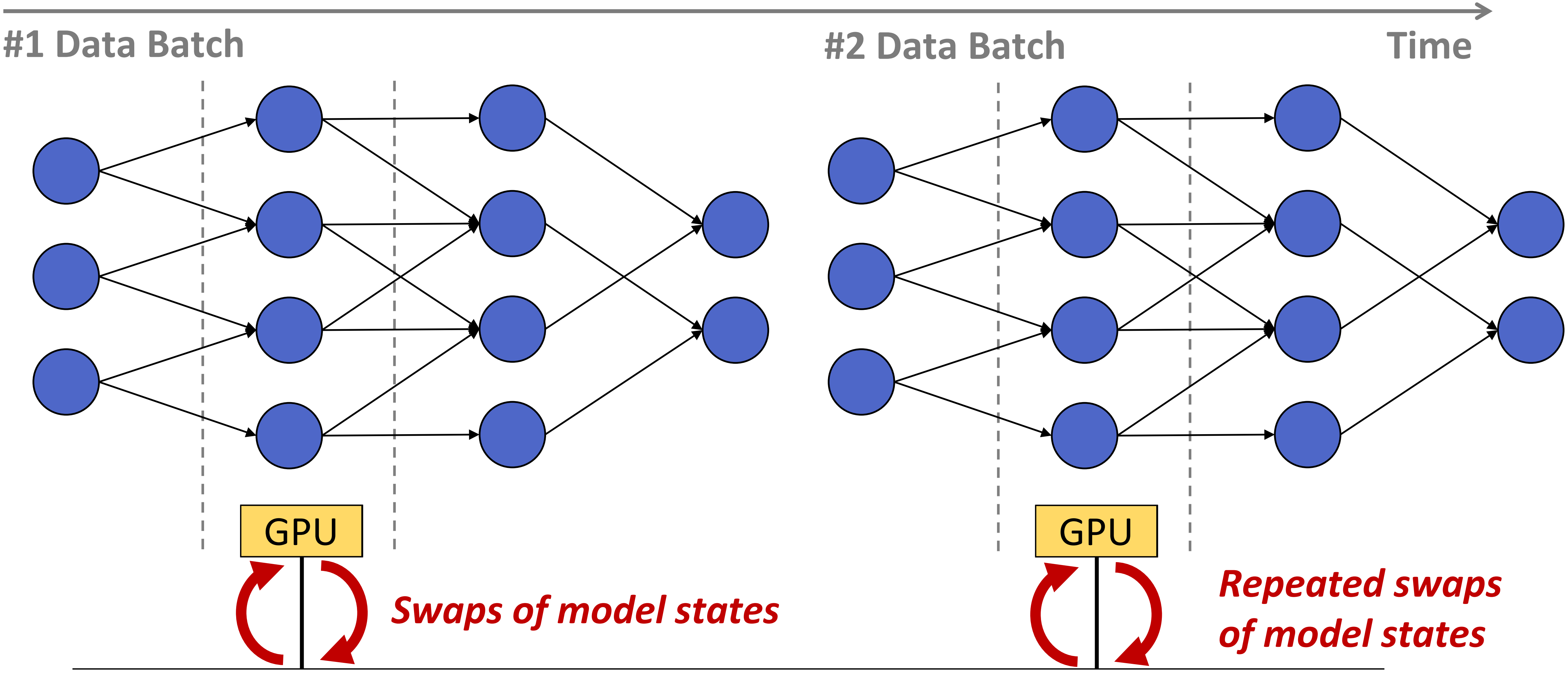
*[vDNN, MICRO'16]
[LMS, SysML'18]
[SwpAdv, ASPLOS'20]
[Sentinel, HPCA'21]

Excessive overhead: Repeated swaps across data batches

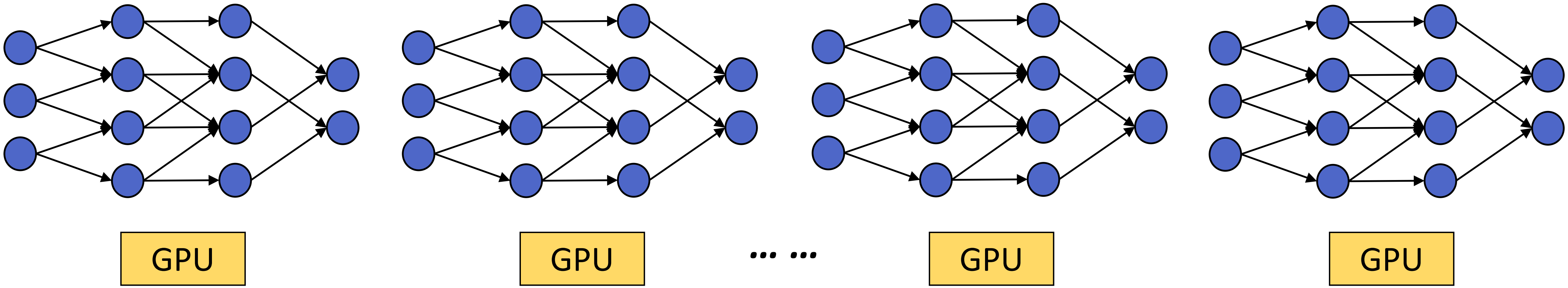
Excessive overhead: Repeated swaps across data batches



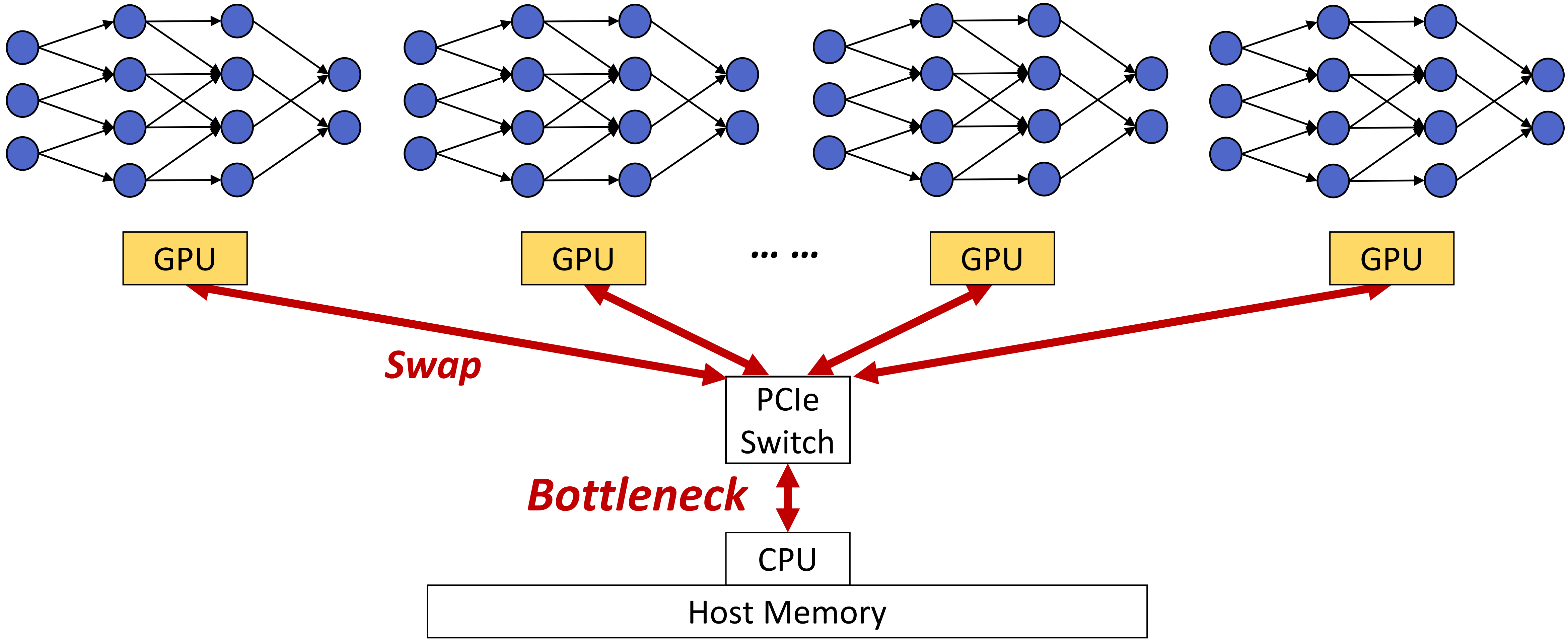
Excessive overhead: Repeated swaps across data batches



Swapping bottleneck: All GPU swaps via a root link

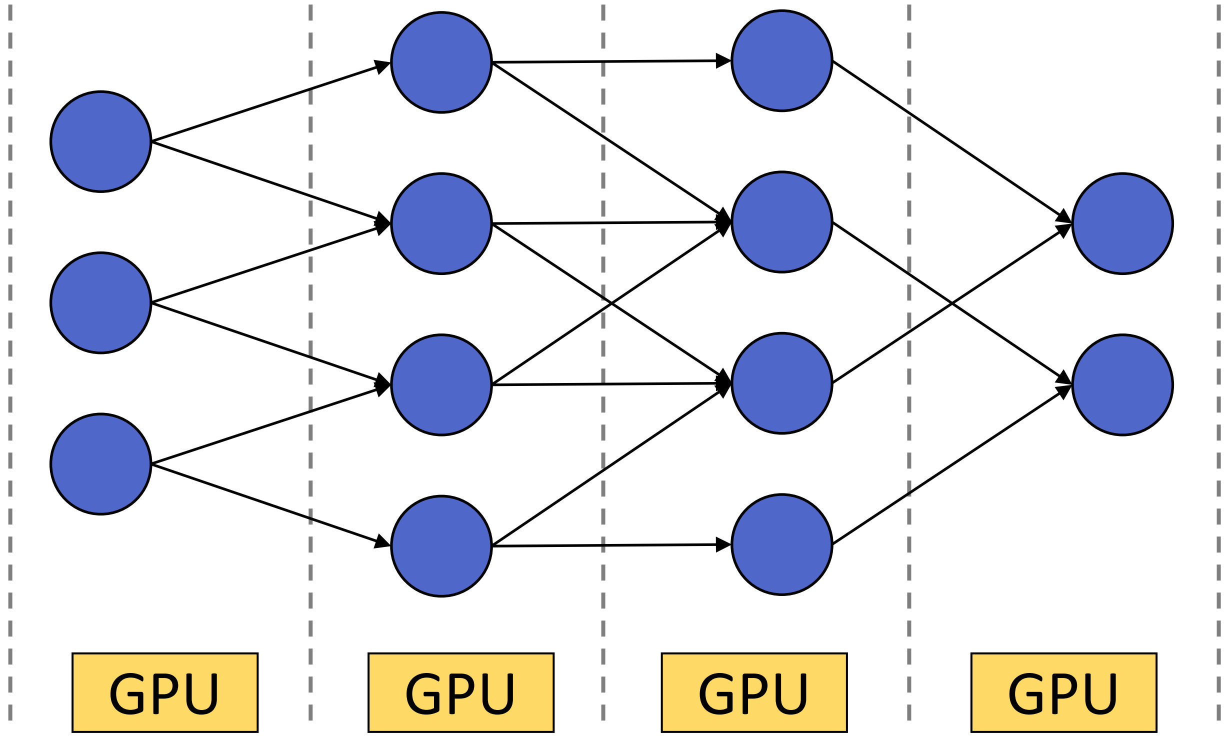


Swapping bottleneck: All GPU swaps via a root link



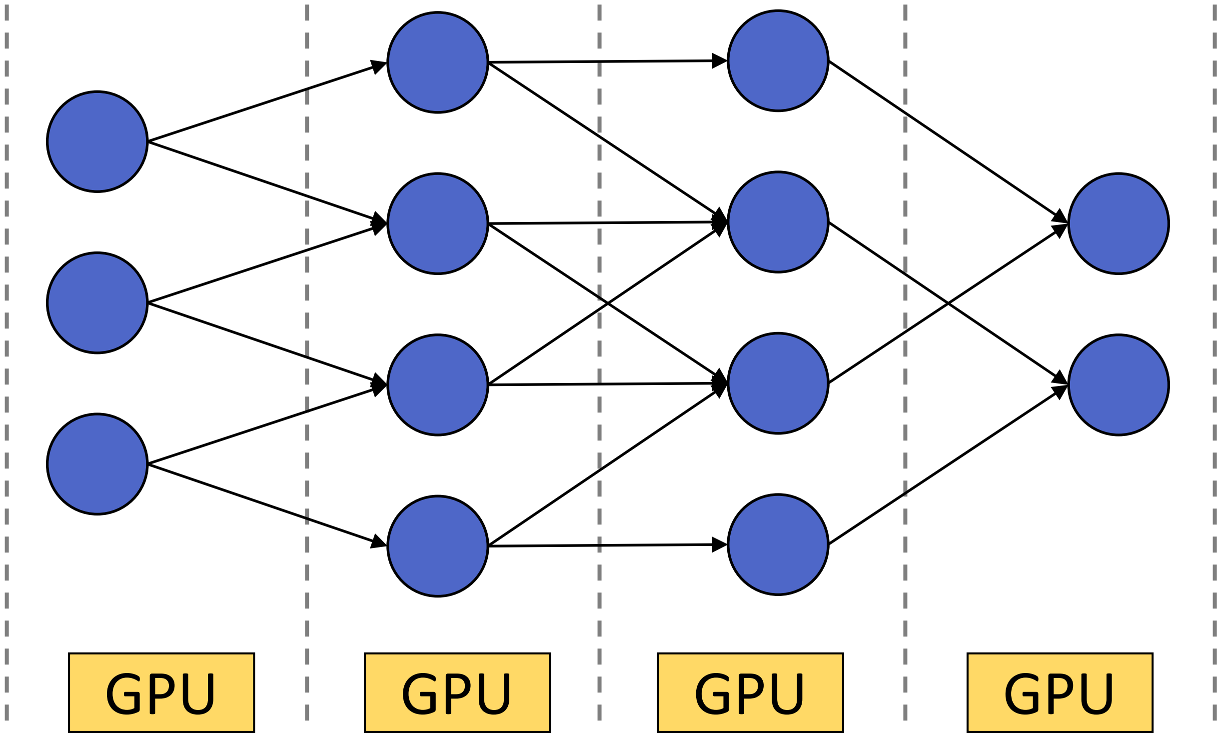
Intra-server Interconnects

Promising technique #2: *Model/Pipeline Parallel Training*



- **Model-parallel training** [Megatron, arXiv'19]
- **Pipeline-parallel training** [GPipe, NeurIPS'19] [PipeDream, SOSP'19]

Promising technique #2: *Model/Pipeline Parallel Training*

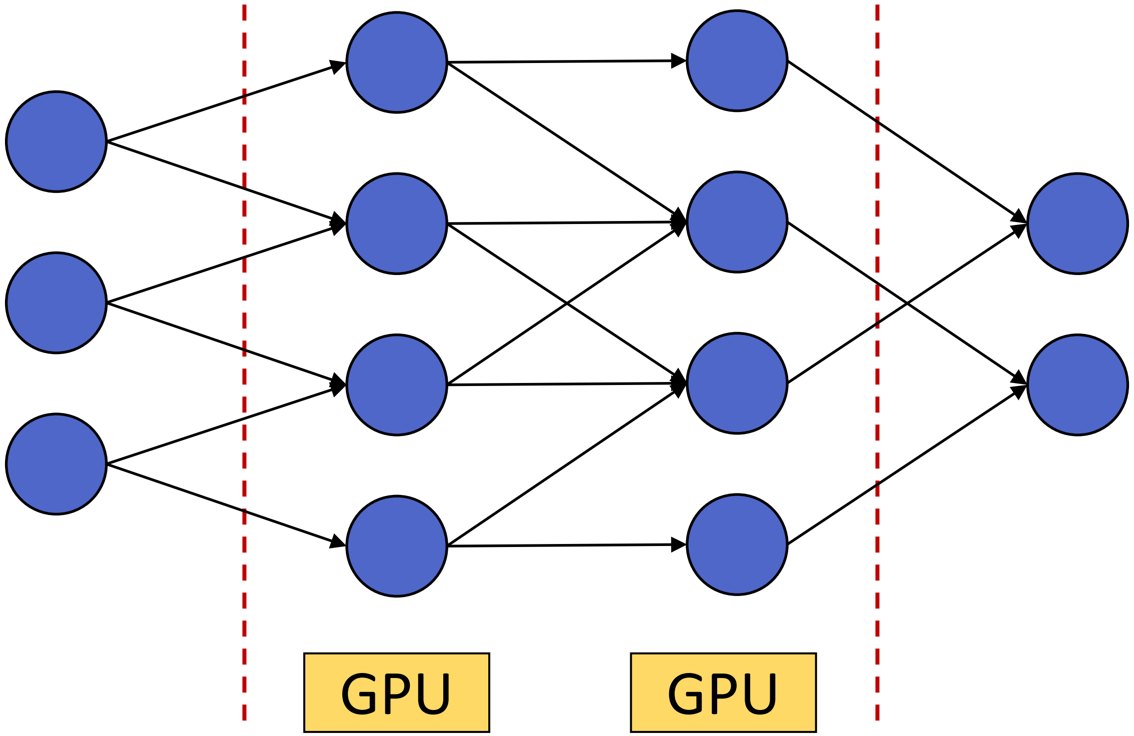


- **Model-parallel training** [Megatron, arXiv'19]
- **Pipeline-parallel training** [GPipe, NeurIPS'19] [PipeDream, SOSP'19]

Great for models that fit collective memory capacity :)

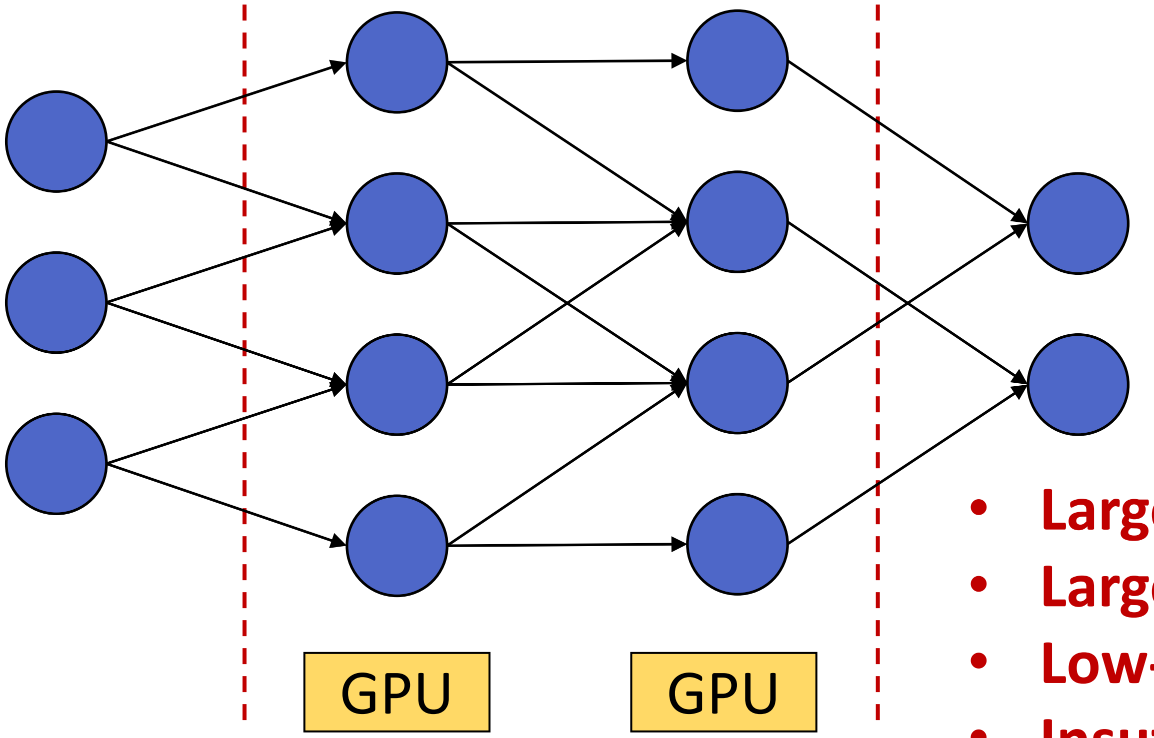
Quite often, on commodity servers:

training memory footprint > collective accelerator memory capacity



Quite often, on commodity servers:

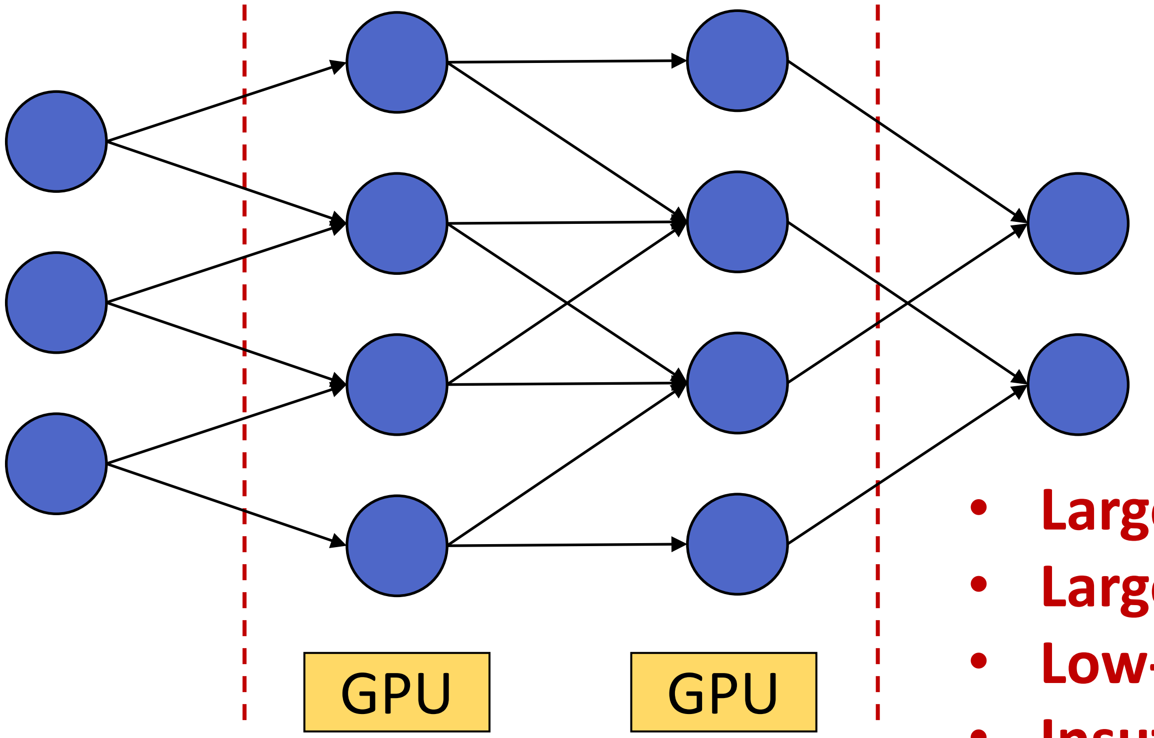
training memory footprint > collective accelerator memory capacity



- **Large Model Size**
- **Large Data Size**
- **Low-End GPUs**
- **Insufficient # of GPUs**

Quite often, on commodity servers:

training memory footprint > collective accelerator memory capacity

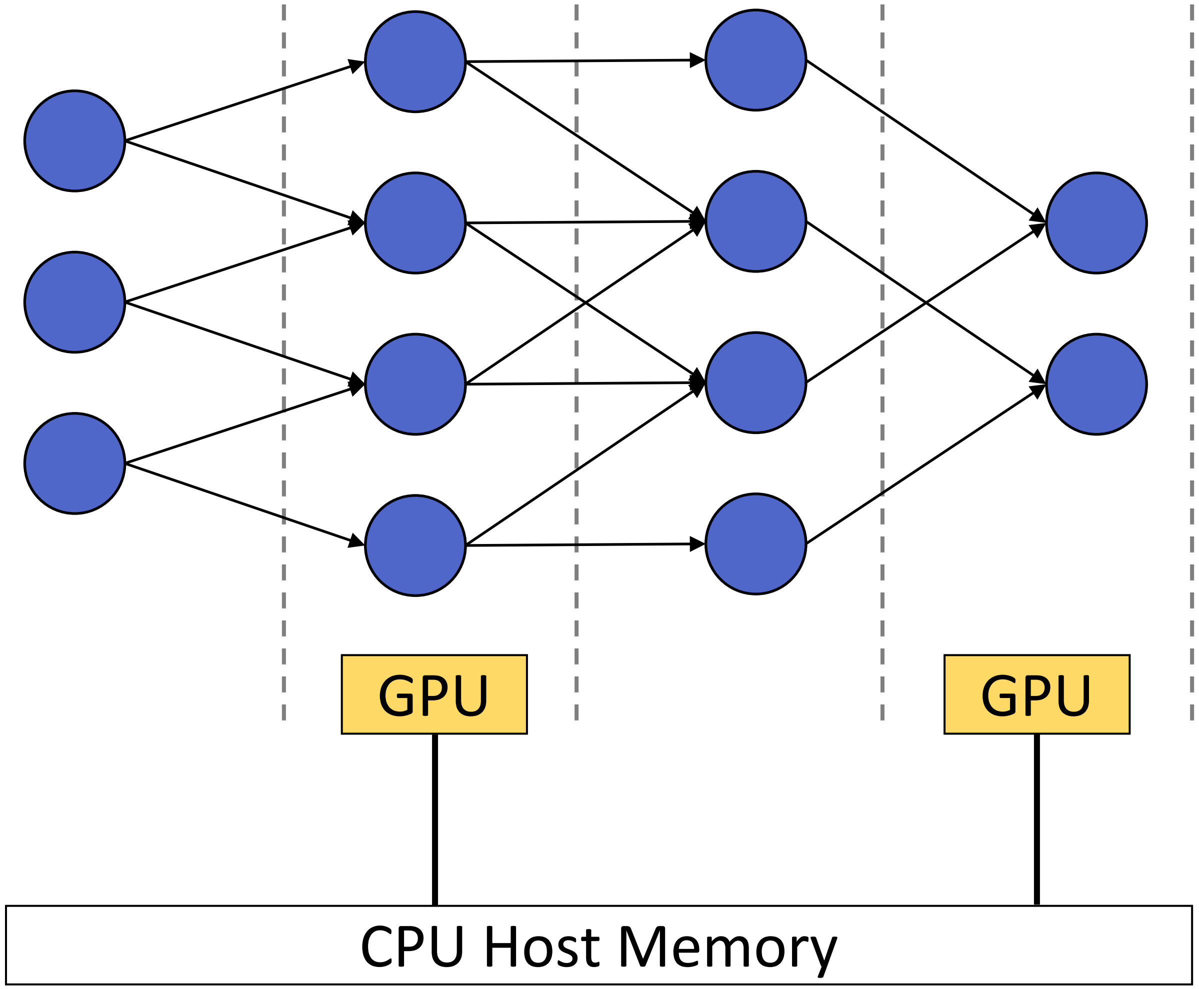


- **Large Model Size**
- **Large Data Size**
- **Low-End GPUs**
- **Insufficient # of GPUs**

How to train large models on commodity servers? efficiently?

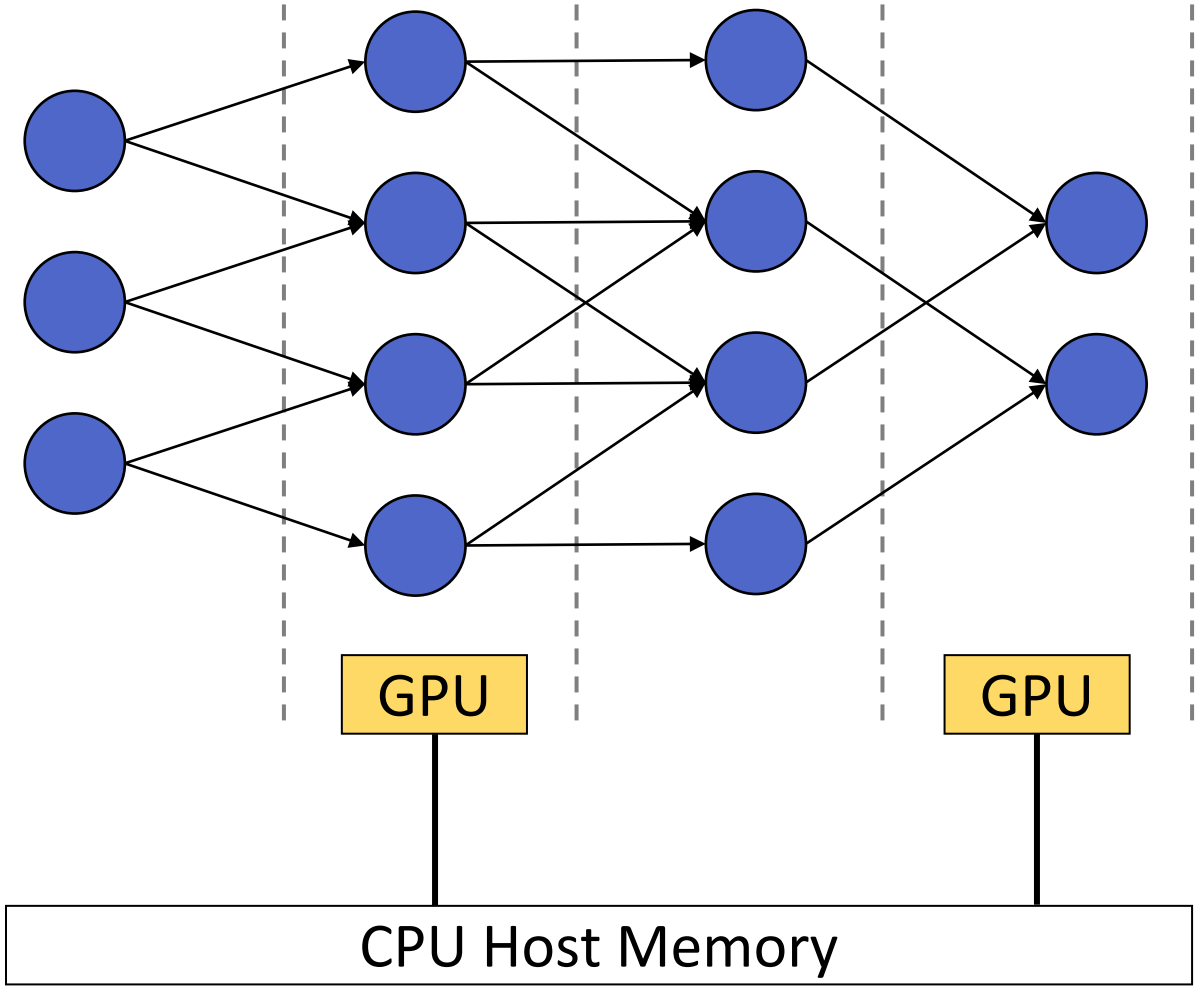
-- Open questions

How about combine existing techniques?



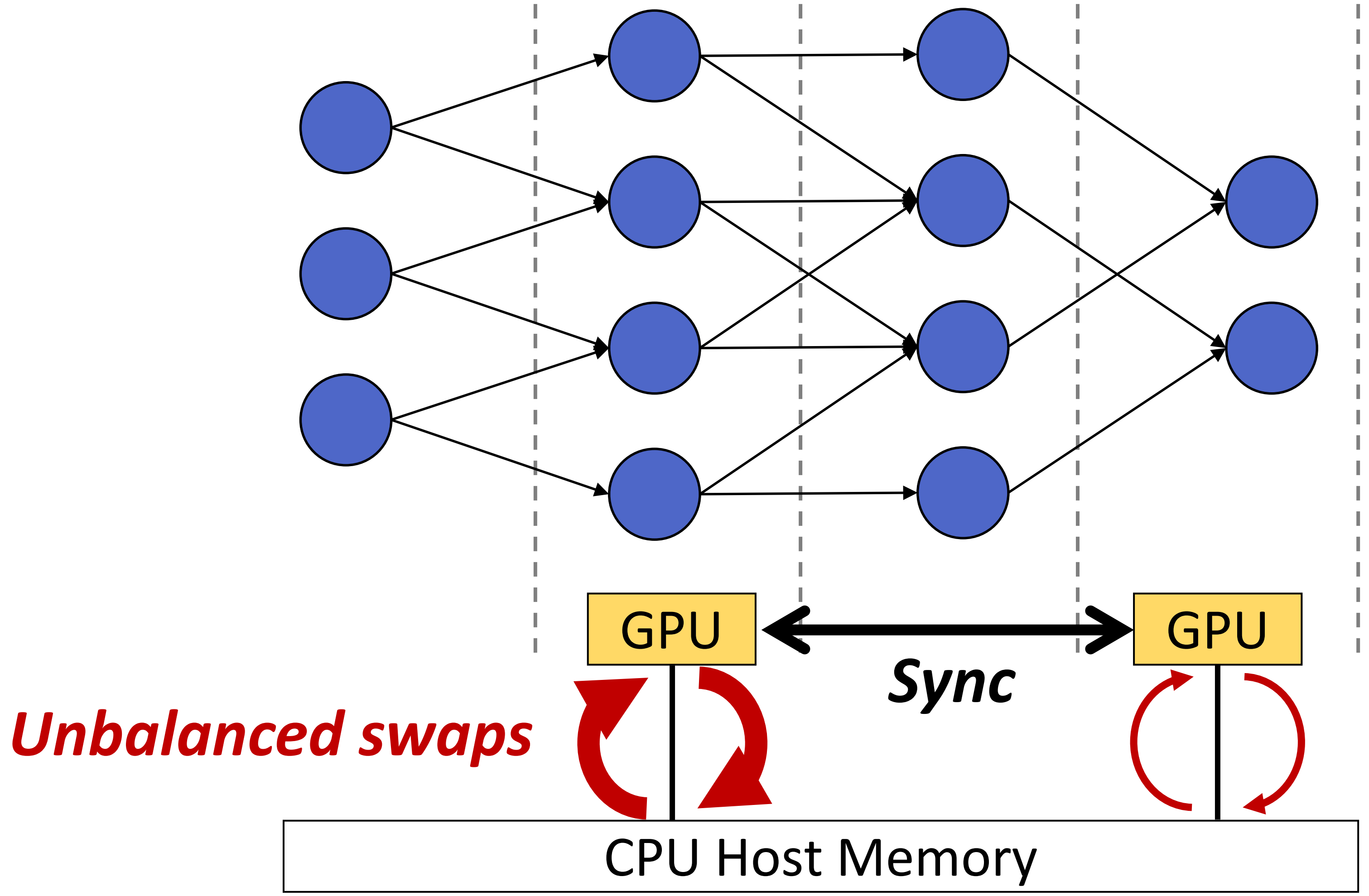
How about combine existing techniques?

-- It is still inefficient



How about combine existing techniques?

-- It is still inefficient



These inefficiencies are hard to solve

- *Excessive swap overhead*
- *CPU-GPU swap bottleneck*
- *Unbalance swaps*

These inefficiencies are hard to solve

- *Excessive swap overhead*
- *CPU-GPU swap bottleneck*
- *Unbalance swaps*

→ **The fundamental limitations in current ML frameworks**

These inefficiencies are hard to solve

- *Excessive swap overhead*
- *CPU-GPU swap bottleneck*
- *Unbalance swaps*

→ The fundamental limitations in current ML frameworks

1. **Coarse-Granularity Scheduling**
   (e.g., PyTorch schedules an entire model to compute the 1st data batch, before the 2nd data batch)

These inefficiencies are hard to solve

- *Excessive swap overhead*
- *CPU-GPU swap bottleneck*
- *Unbalance swaps*

→ The fundamental limitations in current ML frameworks

1. **Coarse-Granularity Scheduling**
   (e.g., PyTorch schedules an entire model to compute the 1st data batch, before the 2nd data batch)
2. **Early Binding to Fixed Devices**
   (e.g., in user's training scripts, bind the 1st part of model to 1st GPU and the 2nd part to 2nd GPU)

These inefficiencies are hard to solve

- *Excessive swap overhead*
- *CPU-GPU swap bottleneck*
- *Unbalance swaps*

→ The fundamental limitations in current ML frameworks

1. **Coarse-Granularity Scheduling**
   (e.g., PyTorch schedules an entire model to compute the 1st data batch, before the 2nd data batch)
2. **Early Binding to Fixed Devices**
   (e.g., in user's training scripts, bind the 1st part of model to 1st GPU and the 2nd part to 2nd GPU)

→ ***Not flexible & poor resource utilization***

Our proposal – Harmony

Our proposal – Harmony

Key Idea

- **Decompose** everything (data, model, operations) into small tasks

Our proposal – Harmony

Key Idea

- **Decompose** everything (data, model, operations) into small tasks
- Schedule tasks at a **fine granularity**

Our proposal – Harmony

Key Idea

- **Decompose** everything (data, model, operations) into small tasks
- Schedule tasks at a **fine granularity**
- **Late-bind** tasks’ compute and their swaps onto hardware

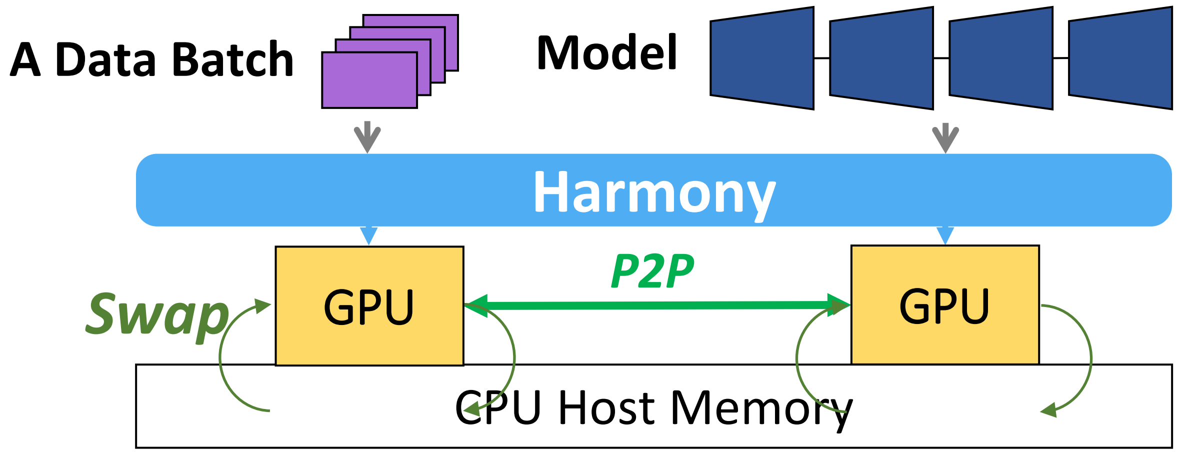
Our proposal – Harmony

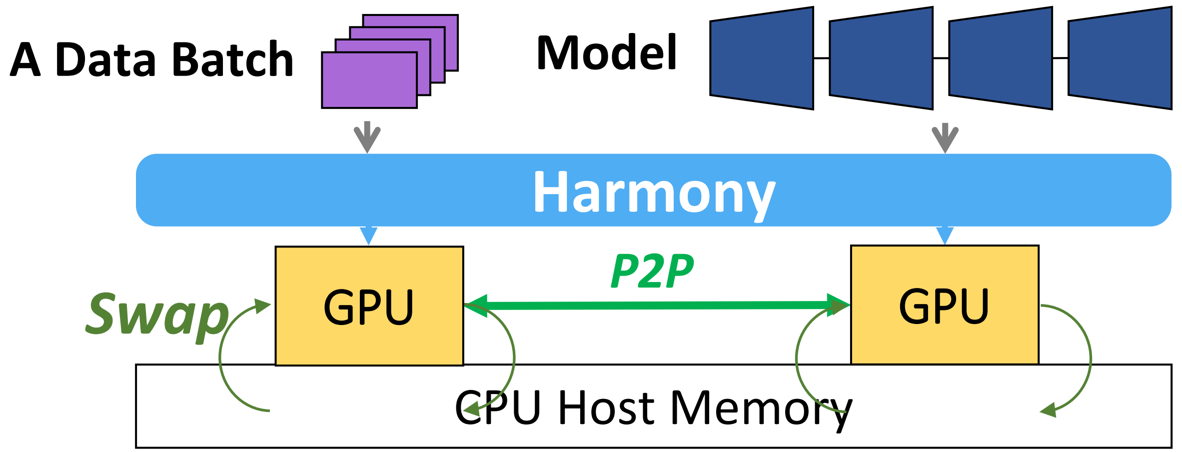
Key Idea

- **Decompose** everything (data, model, operations) into small tasks
- Schedule tasks at a **fine granularity**
- **Late-bind** tasks' compute and their swaps onto hardware

Goal

- Maximize **system efficiency** for training large models

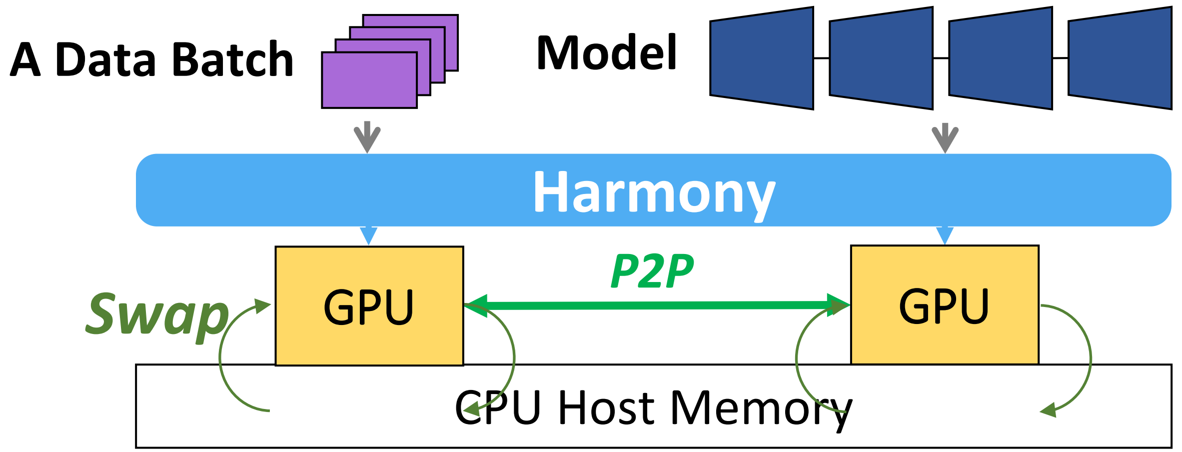

A Data Batch
Model
Harmony
P2P
Swap
GPU
GPU
CPU Host Memory



Four Principles

- **Minimize memory swaps**

  (e.g., *group* tasks to reuse shared states in memory)

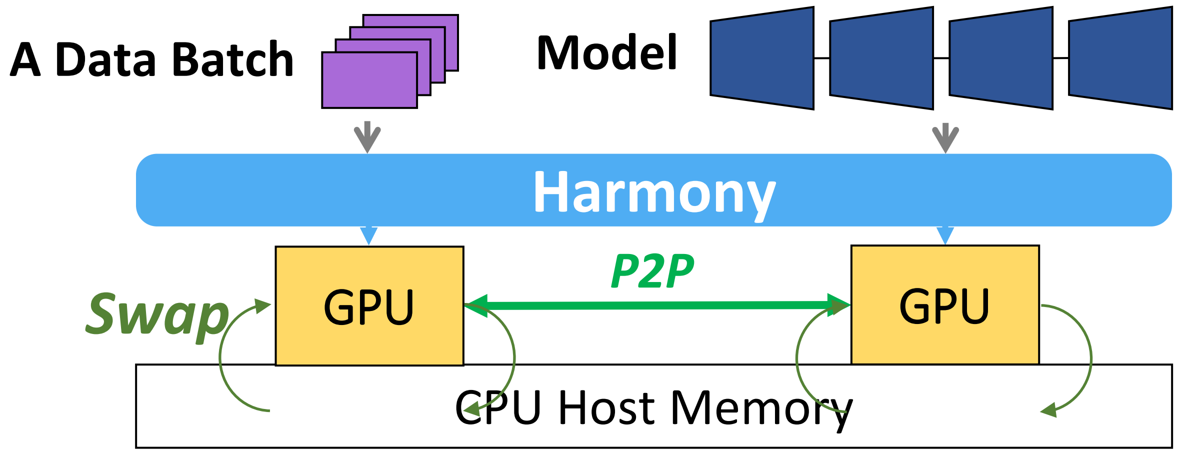


Four Principles

- **Minimize memory swaps**

  (e.g., *group* tasks to reuse shared states in memory)
- **Schedule tasks just-in-time**

  (e.g., *jit-schedule* a task before its input getting swap-out)



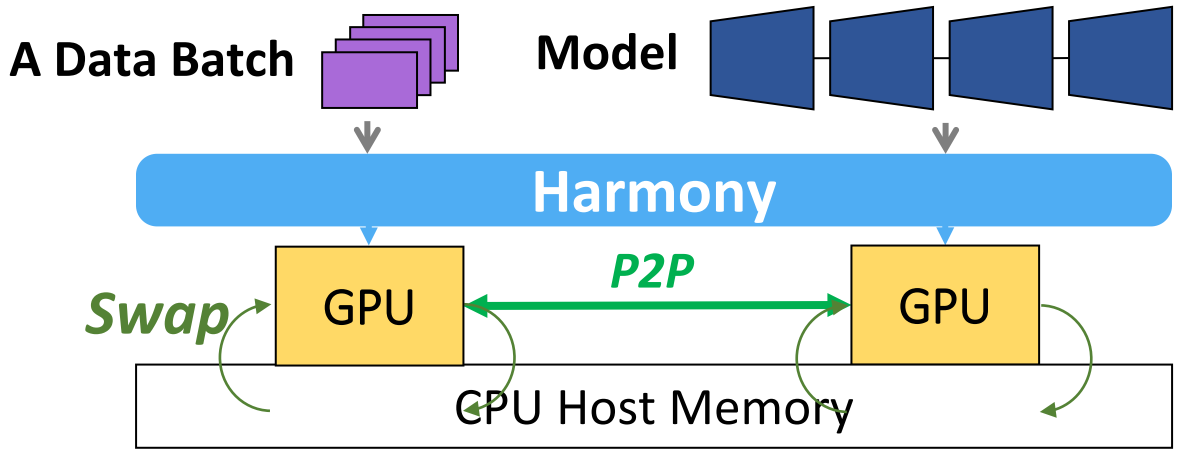
Four Principles

- **Minimize memory swaps**

  (e.g., *group* tasks to reuse shared states in memory)
- **Schedule tasks just-in-time**

  (e.g., *jit-schedule* a task before its input getting swap-out)
- **Swap over fast peer-to-peer links**

  (e.g., *move* CPU-GPU swaps to P2P transfers)



Four Principles

- **Minimize memory swaps**
  (e.g., *group* tasks to reuse shared states in memory)
- **Schedule tasks just-in-time**
  (e.g., *jit-schedule* a task before its input getting swap-out)
- **Swap over fast peer-to-peer links**
  (e.g., *move* CPU-GPU swaps to P2P transfers)
- **Balance load**
  (e.g., *pack* tasks for similar compute & swap loads)

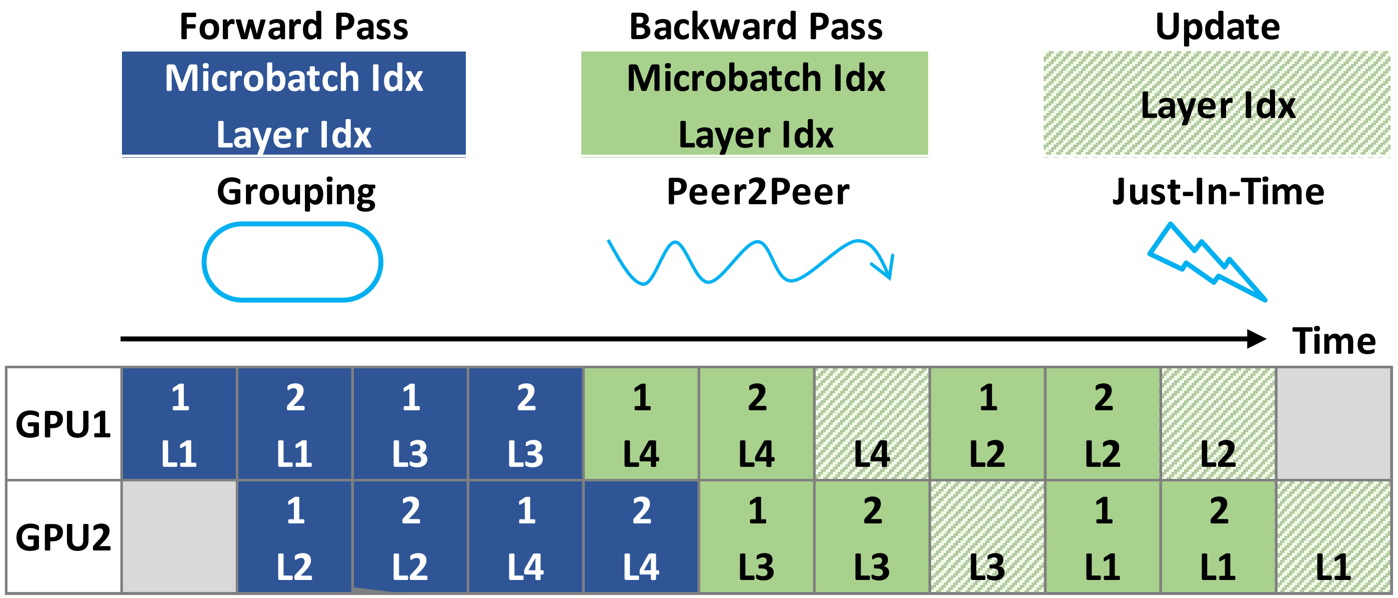
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*

Forward Pass	Backward Pass	Update
Microbatch Idx Layer Idx	Microbatch Idx Layer Idx	Layer Idx

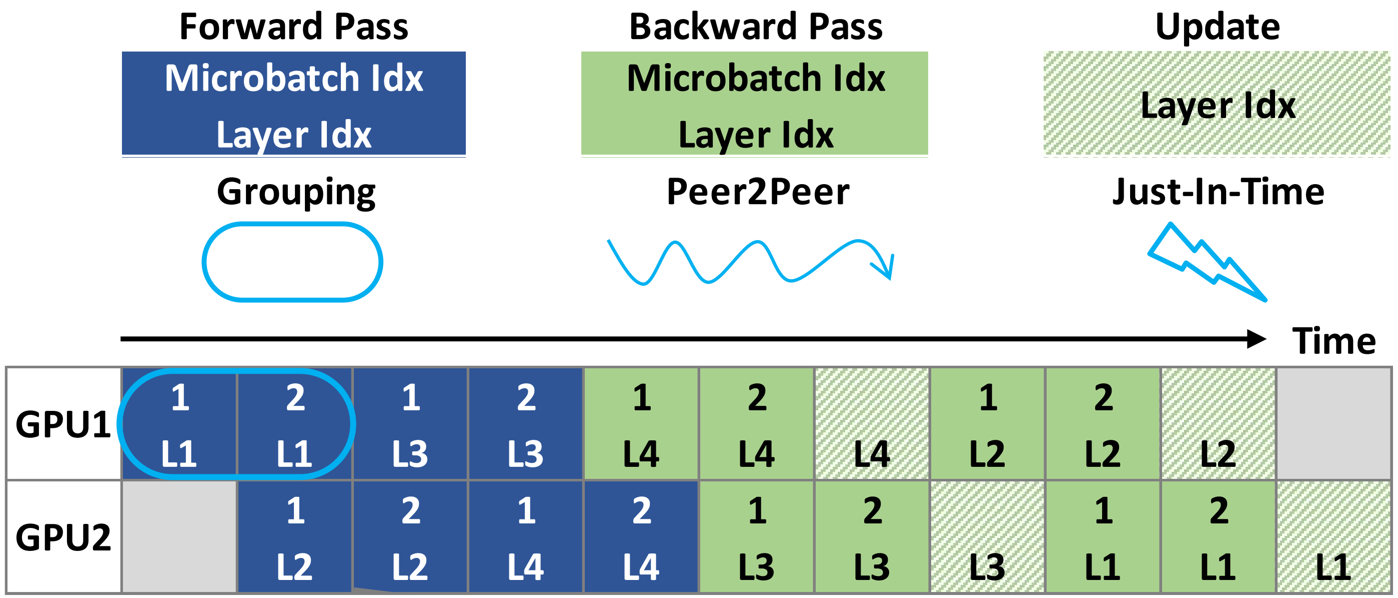
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*

Forward Pass	Backward Pass	Update
Microbatch Idx Layer Idx	Microbatch Idx Layer Idx	Layer Idx
Grouping	Peer2Peer	Just-In-Time

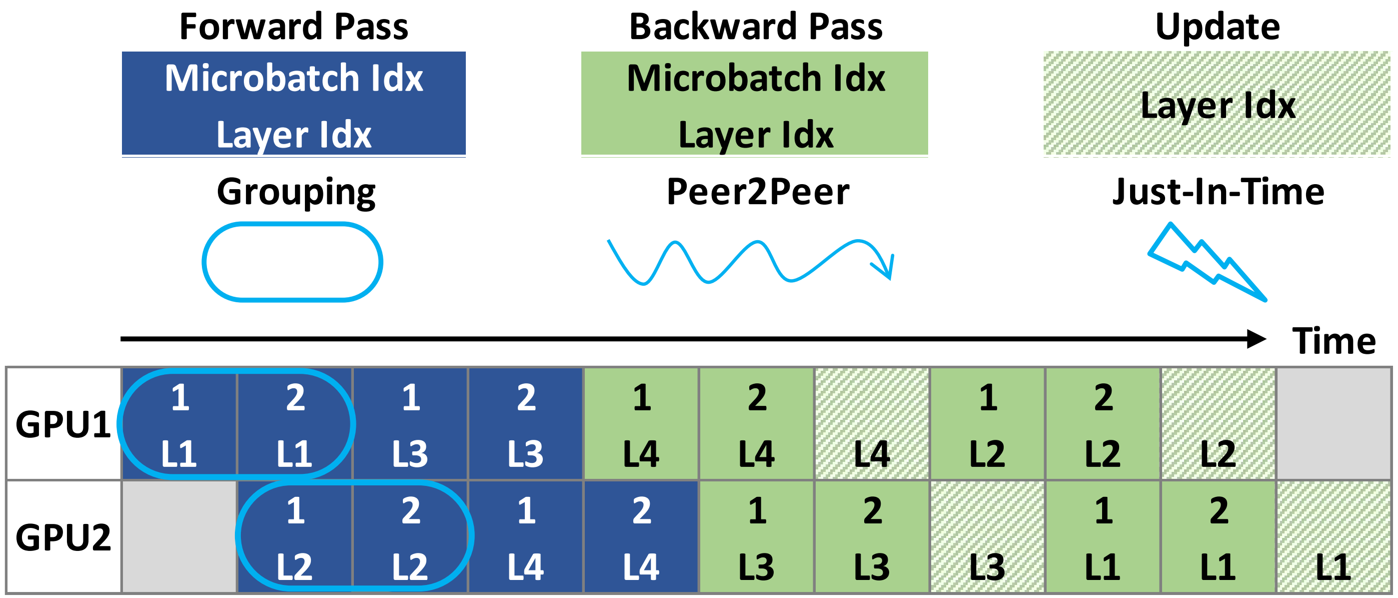
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



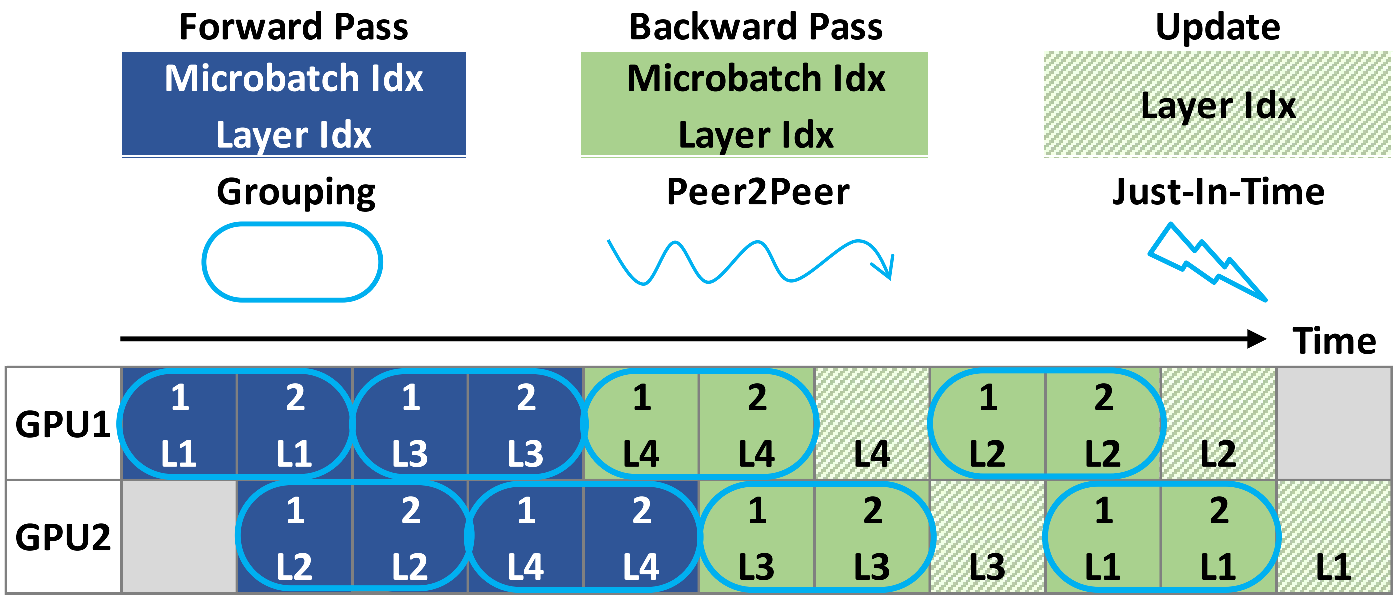
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



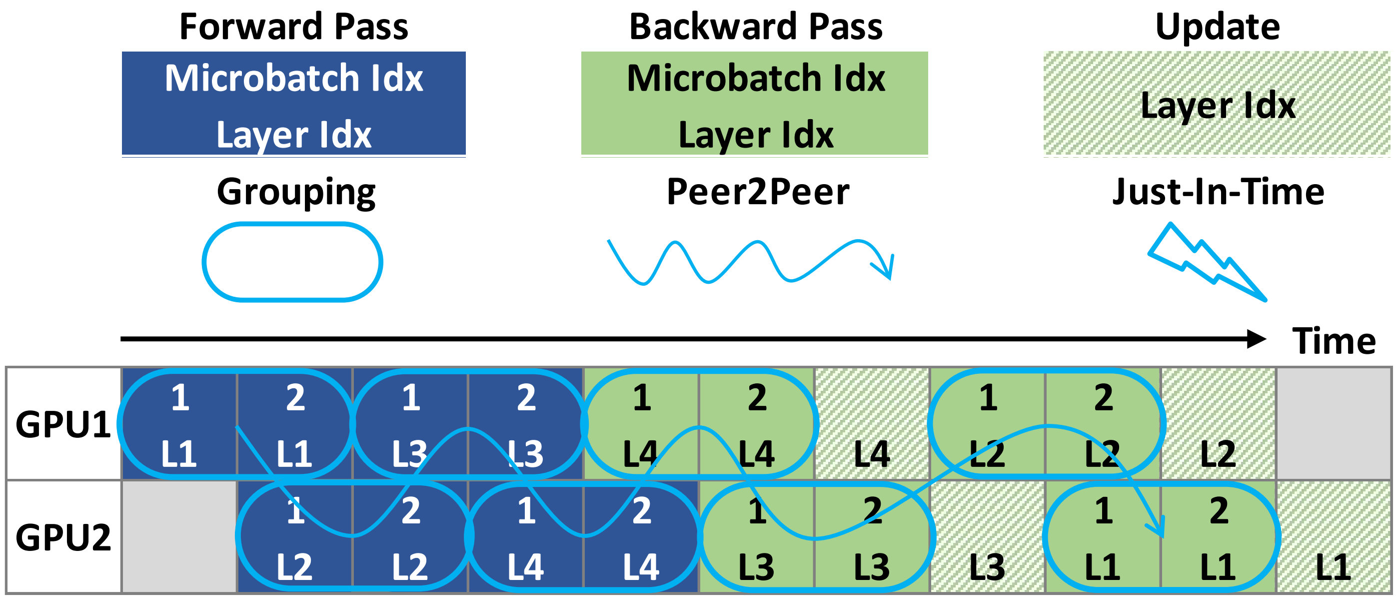
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



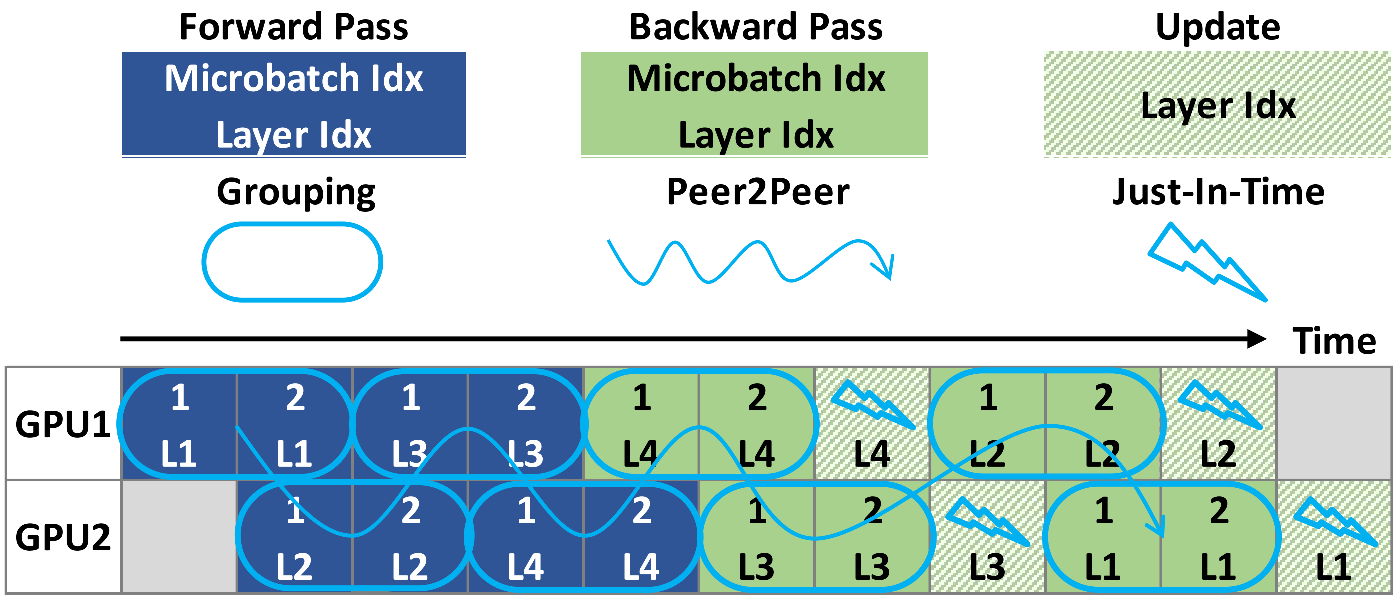
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



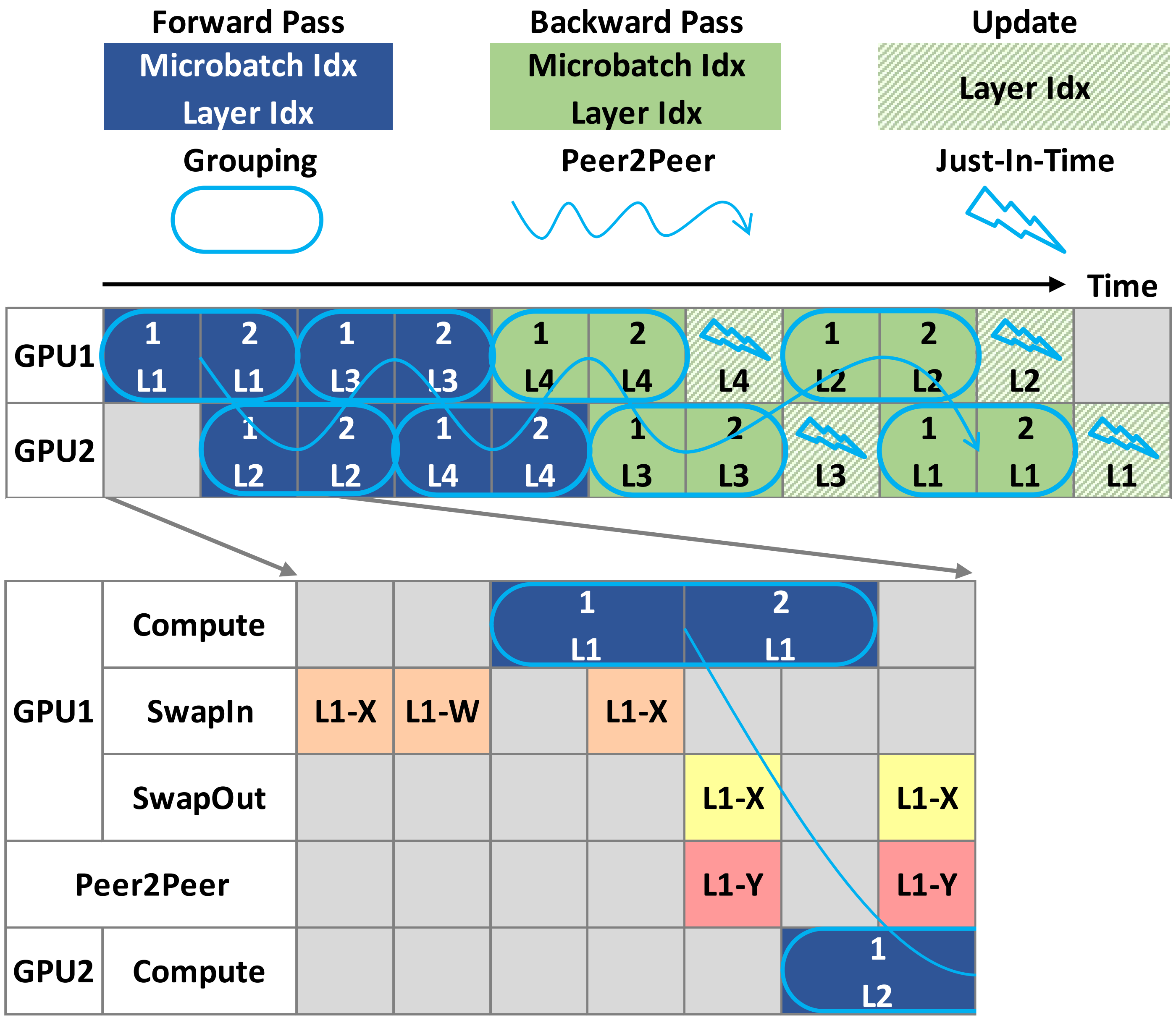
A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



A toy example: *training 4-layer DNN on 2 GPUs with Harmony*



More of Harmony in the paper :)

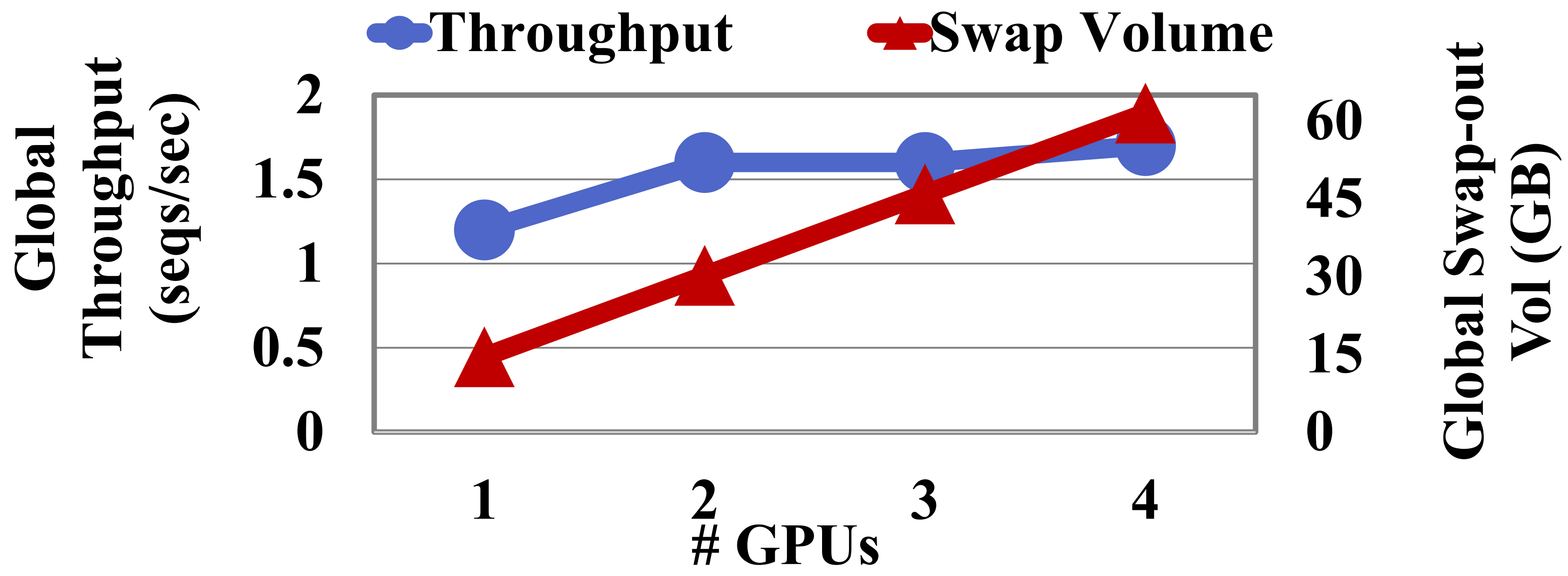
- Single-accelerator abstraction for multi-GPU training
- Virtualization of different parallel training techniques
- Analytical evaluation
- Multi-machine training
- Memory-performance trade-offs
- Feasibility for end-to-end training on modest deployments

Conclusion

- Large model training can also be for the "masses"!
- Large model training requires huge accelerator memory.
- Memory virtualization incurs excessive swap overhead.
- We advocate rethinking how ML frameworks schedule compute and move data for – *efficiently training large models on commodity servers.*

Backup Slides

Data-Parallelism with per-GPU memory virtualization



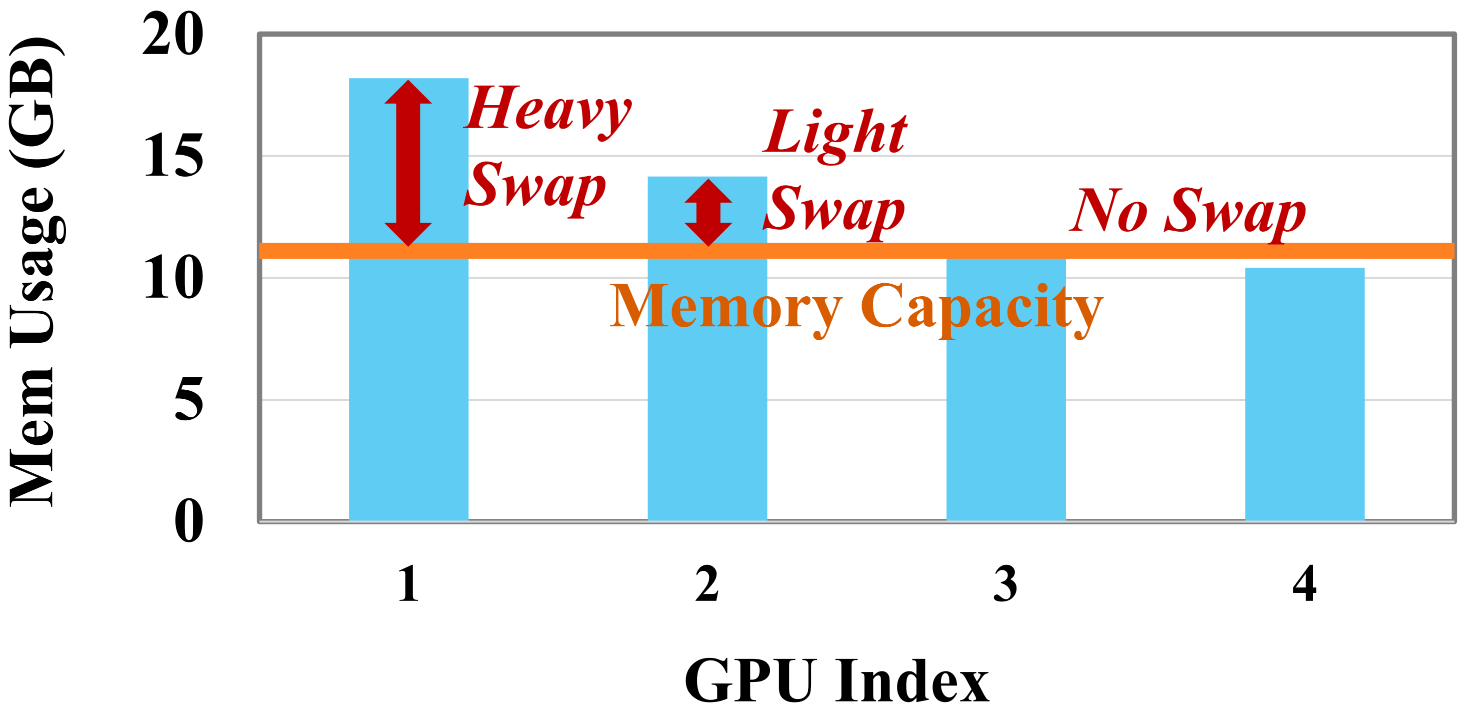
Setting: 4x GTX-1080Ti (11GB) + Bert-Large + Per-GPU batchsize=5 + PyTorch Data Parallel + IBM-LMS

Linearly increased swap volume plagues throughput

(Reason: swap load across replicated models is proportional to GPU count)

* [LMS, SysML'18]

Pipeline-Parallel training with per-GPU memory virtualization



Setting: 4x GTX-1080Ti (11GB) + Bert-Large + MicrobatchSize 32+ PipeDream2BW + IBM-LMS

Unbalanced swap load across GPUs

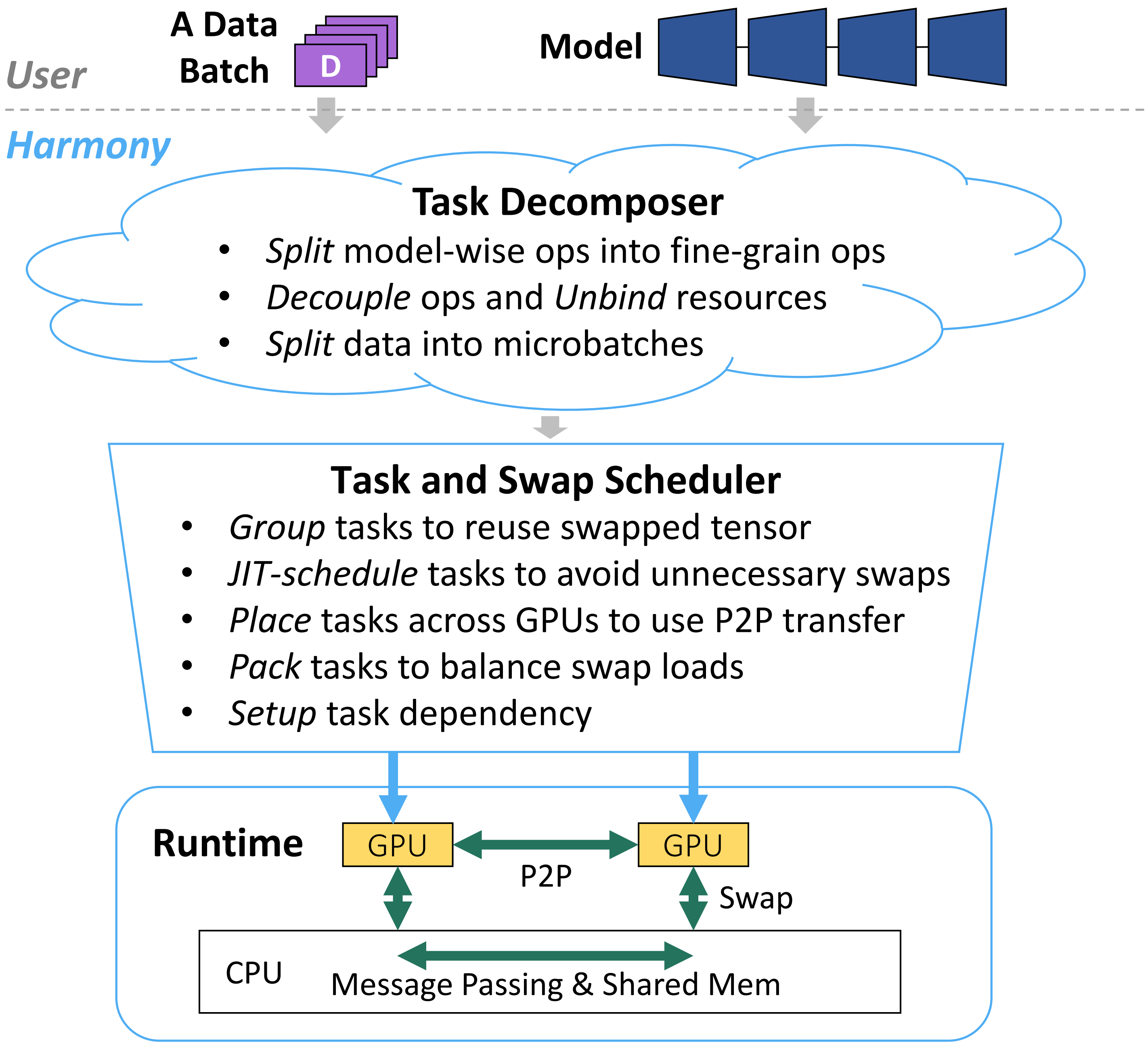
→ The heaviest swap happens on GPU-0 while GPU2-3 has no swap issue

→ Pipeline is always synchronous across all GPUs

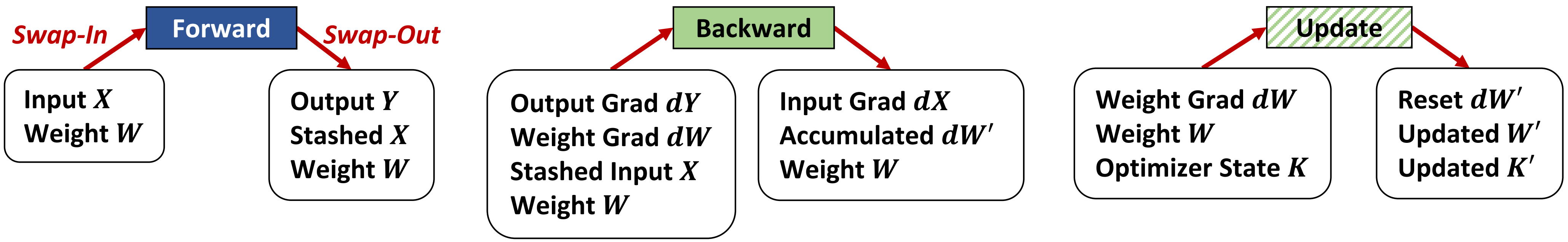
→ The system throughput is bottlenecked by GPU-0

* [LMS, SysML'18]

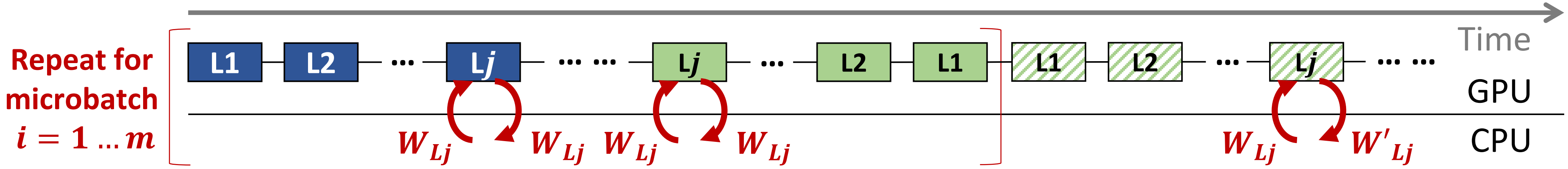
Overview


A Data Batch
D
Model
User
Harmony
Task Decomposer
• Split model-wise ops into fine-grain ops
• Decouple ops and Unbind resources
• Split data into microbatches
Task and Swap Scheduler
• Group tasks to reuse swapped tensor
• JIT-schedule tasks to avoid unnecessary swaps
• Place tasks across GPUs to use P2P transfer
• Pack tasks to balance swap loads
• Setup task dependency
Runtime
GPU
GPU
P2P
Swap
CPU
Message Passing & Shared Mem

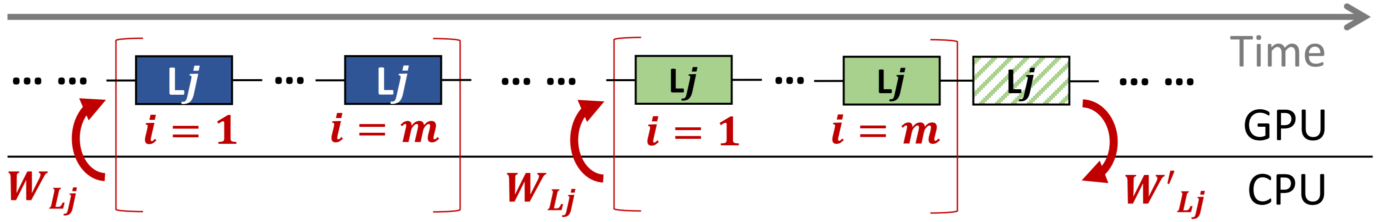
Analyzing the swap load



(a) Swap model.



(b) Swapping of weights for layer Lj in "DP with per-GPU memory virtualization."



(c) Swapping of weights for layer Lj in "Harmony DP."

Analytical comparison

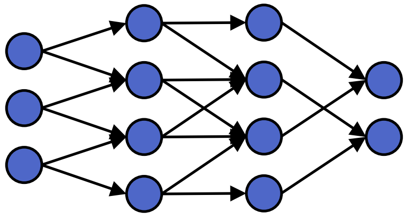
Different Approaches			DP with Per-GPU Swap	Harmony DP (vDP)	PP with Per-GPU Swap	Harmony PP (vPP)
Total Comm. Volume of One Data Batch	Swap Volume (In + Out)	W	$(4M + 2N) \times$	$3N \times$	$(4M + 2) \times$	$3 \times$
		dW	$(2M + 2N) \times$	$0 \times$	$(2M + 2) \times$	$0 \times$
		K	$2N \times$	$2N \times$	$2 \times$	$2 \times$
		X	$2M \times$	$2M \times$	$2M \times$	$2M \times$
		Y & dX	-	$2M \times$	-	-
	P2P Volume	AllReduce dW	$N \times$	$N \times$	-	-
		Send Y & dX	-	-	$\left(\frac{N-1}{R}\right) M \times$	$M \times$
Balanced Memory Usage (CPU + GPU)			Yes	Yes	No (Stashed X across GPUs is $1:2:\ldots:N$)	Yes

Feasibility for end-to-end training on commodity GPU servers

End-to-End Training Timeline

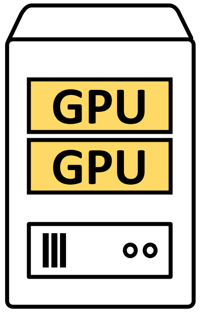
Large model

1. Development & Debugging

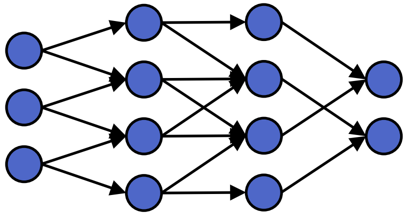


Training FLOPs: up-to 10^{19} *or*
Training Time: several hours/days

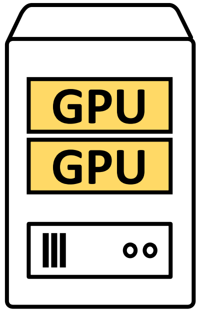
Commodity machine(s) with only GTX GPU



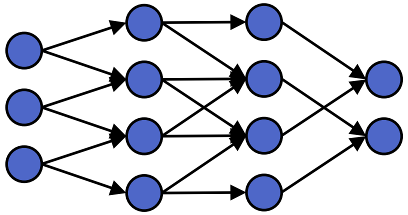
2. Pre-training from Scratch



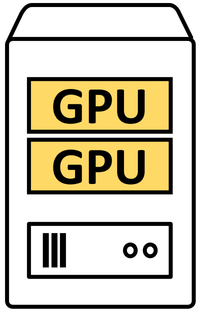
Training FLOPs: beyond 10^{23} *or*
Training Time: months/years



3. Fine-tuning to Best Accuracy



Training FLOPs: up-to 10^{19} *or*
Training Time: several hours/days



Estimated Training Time

Model	Pre-training		Fine-Tuning Wikitext-103 for 5 epoch			Fine-Tuning GLUE for 3 epoch		
	# FLOP	Single 1080Ti Time (days)	# Tokens	# FLOP	Single 1080Ti Time (days)	# Tokens	# FLOP	Single 1080Ti Time (days)
BERT-Large	5.33E+20	581	5.1E+08	1.24E+18	1	6.09E+08	1.49E+18	2
GPT-2	2.38E+21	2,592	5.1E+08	6.30E+18	7	6.09E+08	7.56E+18	8
T5-11B	3.30E+22	36,002	5.1E+08	1.35E+19	15	6.09E+08	1.62E+19	18
GPT-3	3.14E+23	342,564	5.1E+08	7.16E+20	781	6.09E+08	8.59E+20	937