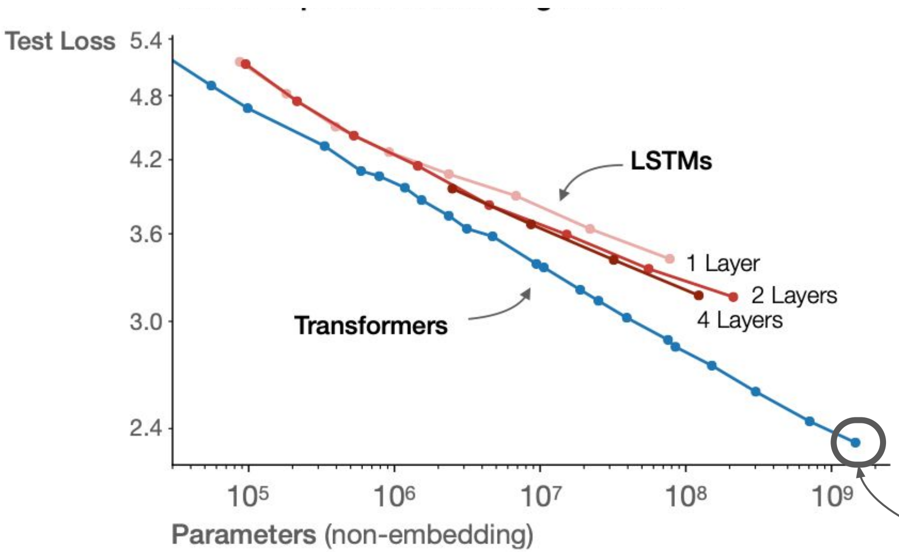
Machine Learning Systems Design

Lecture 5: Scaling Up Training

Stanford University

CS 329S | Karan Goel

Motivation



170 billion

Scale!

Larger models + more data
→ stronger performance

Credit: Scaling Laws for Neural Language Models

OpenAI recently published GPT-3, the largest language model ever trained. GPT-3 has 175 billion parameters and would require 355 years and $4,600,000 to train - even with the lowest priced GPU cloud on the market.[1]

need distributed training!

Credit: Chuan Li, https://lambdalabs.com/blog/demystifying-gpt-3/

Main Ingredients

- Parameters
- Compute
- Data

Main Bottleneck: GPU Memory

Model Training

1. Load the model
2. Load in a minibatch of data
3. Compute forward pass
4. Perform backpropagation

Credit: NVIDIA News

🚀 Parameters: Pre-training with Self-Supervision

BERT (2018) → T5 (2019) → GPT-3 (2020)

BERT (2018)		T5 (2019)		GPT-3 (2020)	
BERT-Base	110M	T5-Small	60M	GPT-3 Small	125M
BERT-Large	**340M**	T5-Base	220M	GPT-3 Medium	350M
		T5-Large	770M	GPT-3 Large	760M
		T5-11B	**11B**	**GPT-3**	**175B**

🚀 Parameters: Pre-training with Self-Supervision

Switch Transformer (2021): 1 Trillion

T5-11B	11B
GPT-3	175B

Storing Parameters

Parameter Representation

32 bit float (float32, fp32)

16 bit float (float16, fp16)

8 bit int unsigned (uint8)

1 B parameters @ fp16
= 10^9 x 2 bytes
= 1.86 GB

175B parameters @ fp16
= 175 x 10^9 x 2 bytes
= 316.2 GB

Note: some parameters must be in float32 for numerical stability

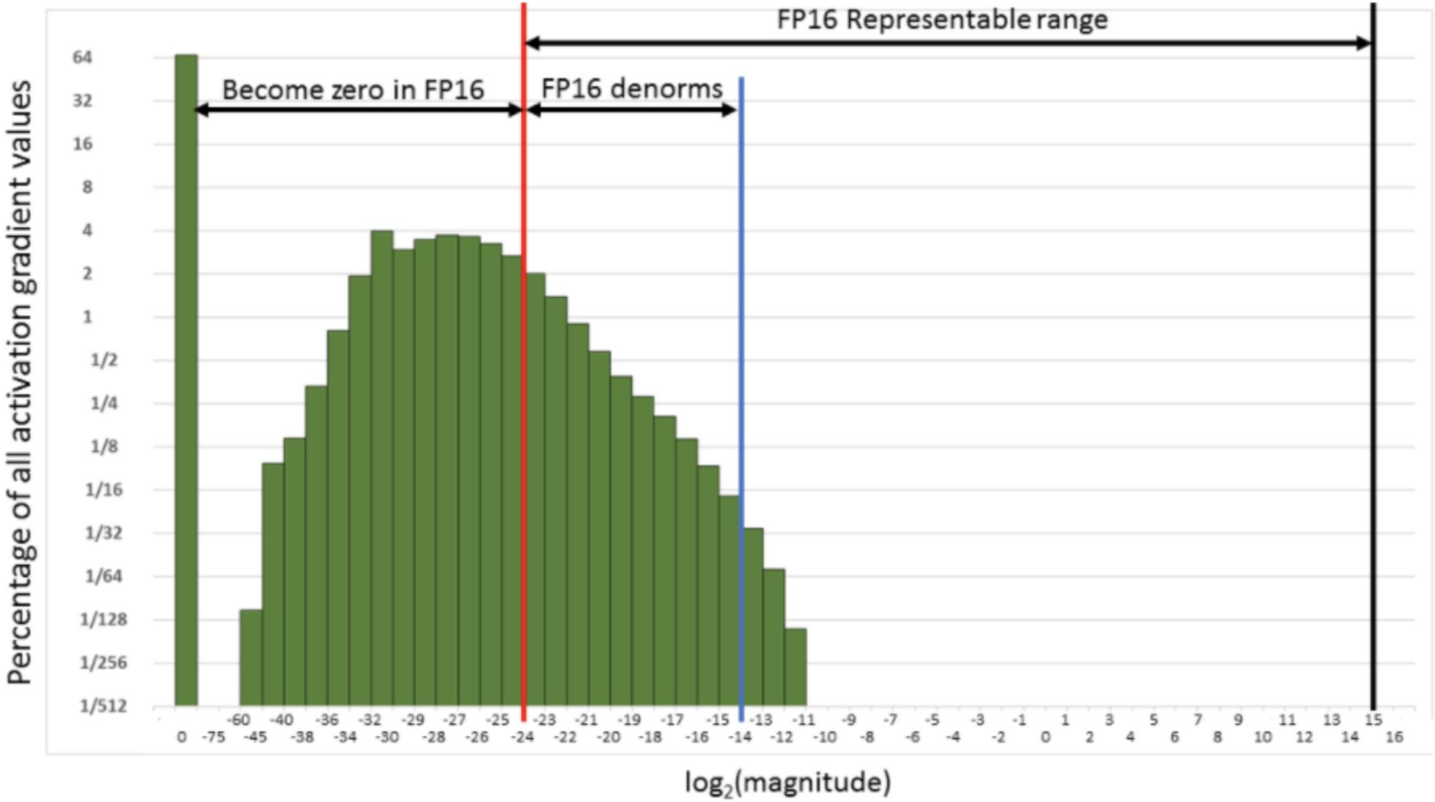
Monday, November 16, 2020

NVIDIA today unveiled the NVIDIA® A100 80GB GPU

80 GB < 316.2 GB

Credit: NVIDIA News

Mixed-Precision Training: Loss Scaling

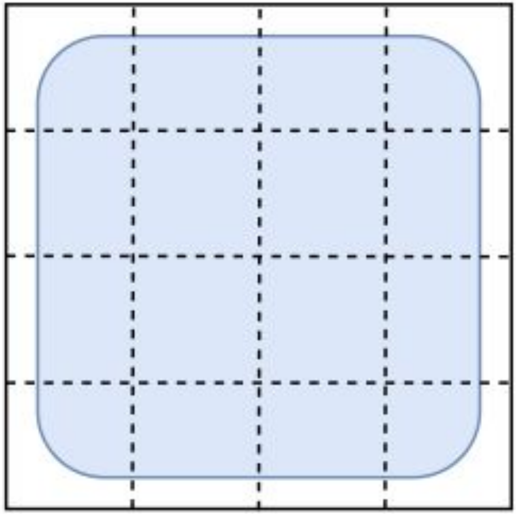


MIXED PRECISION TRAINING (Narang et al., ICLR 2018)

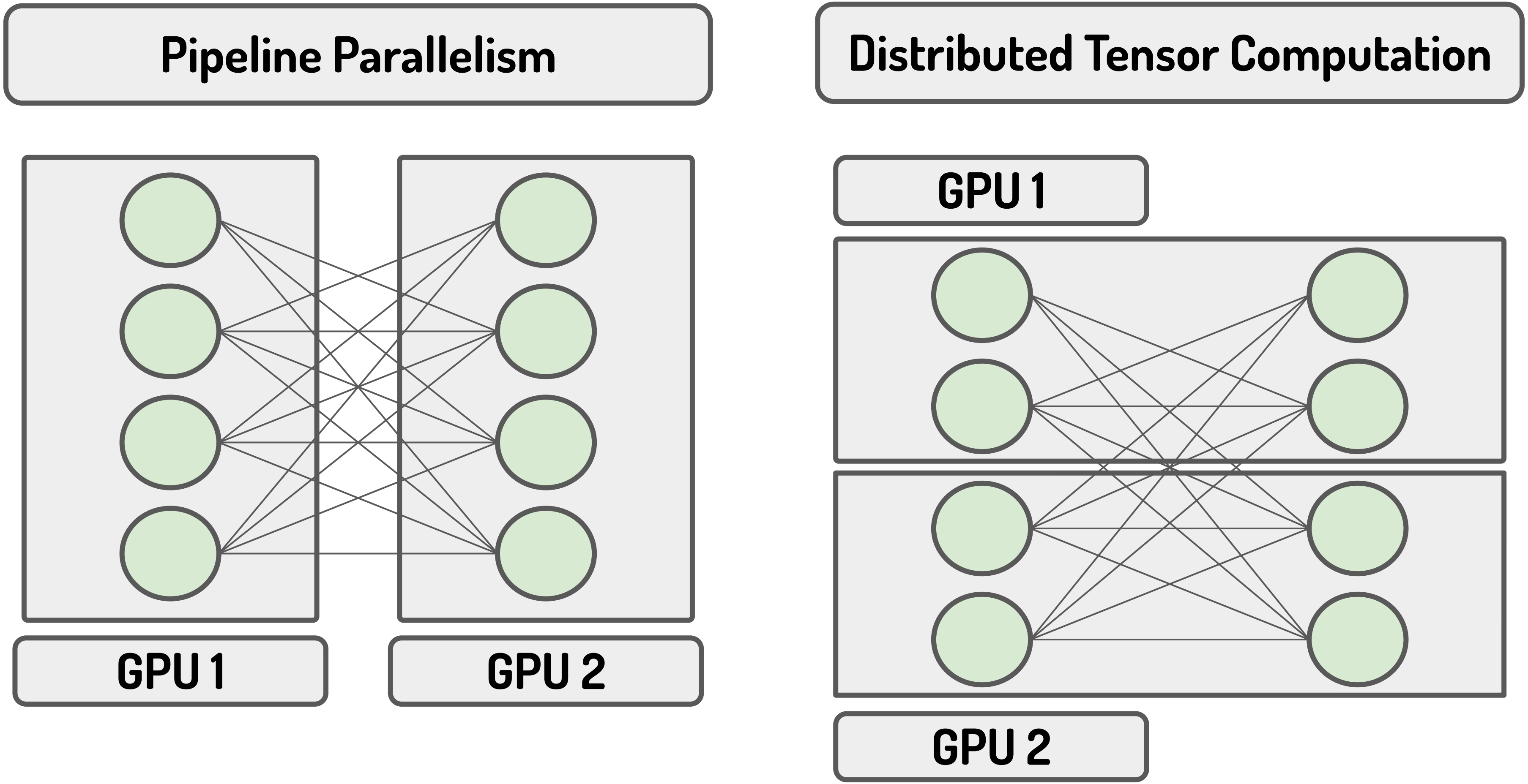
Solution: Model Parallelism for Large Model Training

split the model across devices

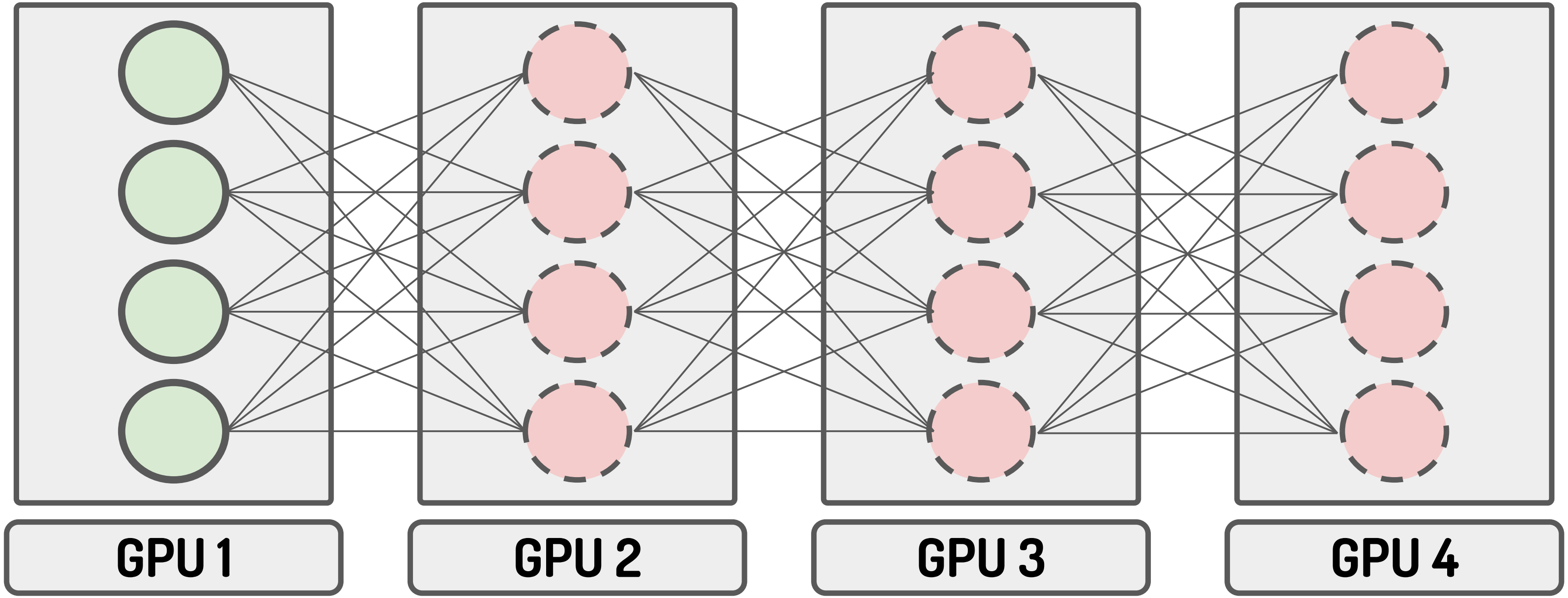
each device runs a fragment of the model



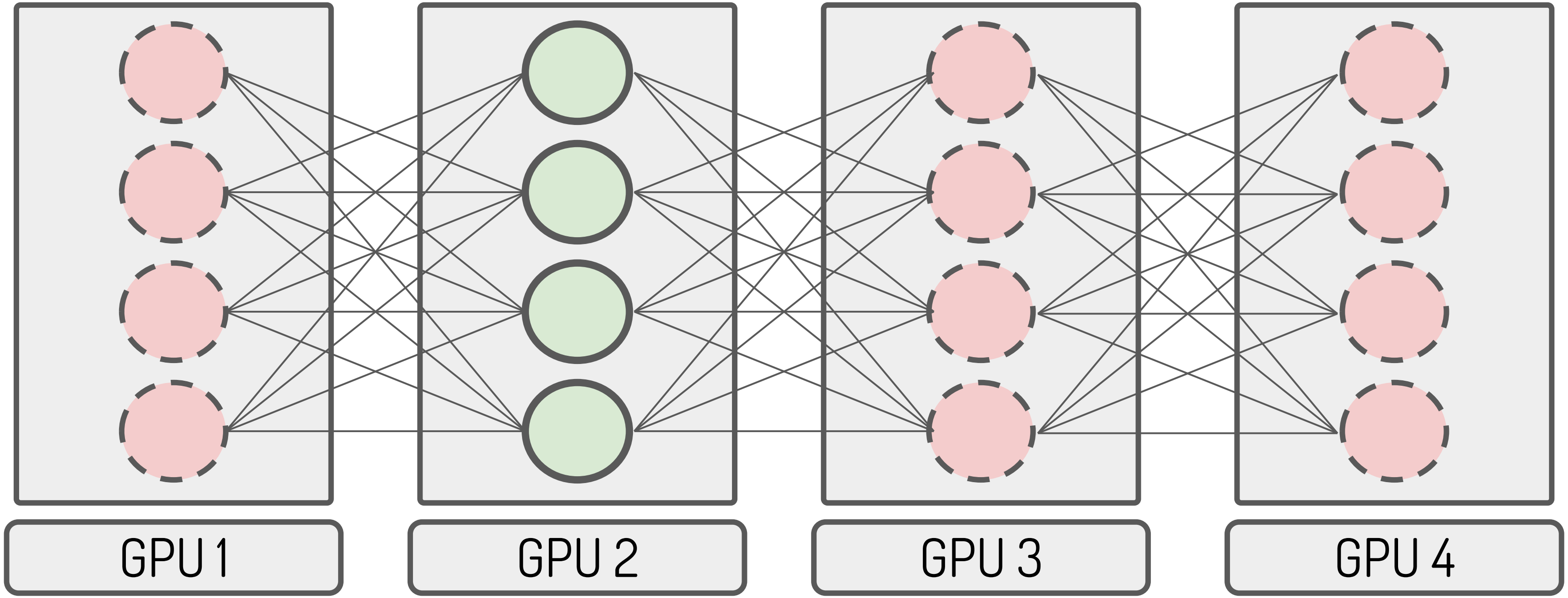
Credit: Fedus et al. (Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity


Pipeline Parallelism
GPU 1
GPU 2
Distributed Tensor Computation
GPU 1
GPU 2

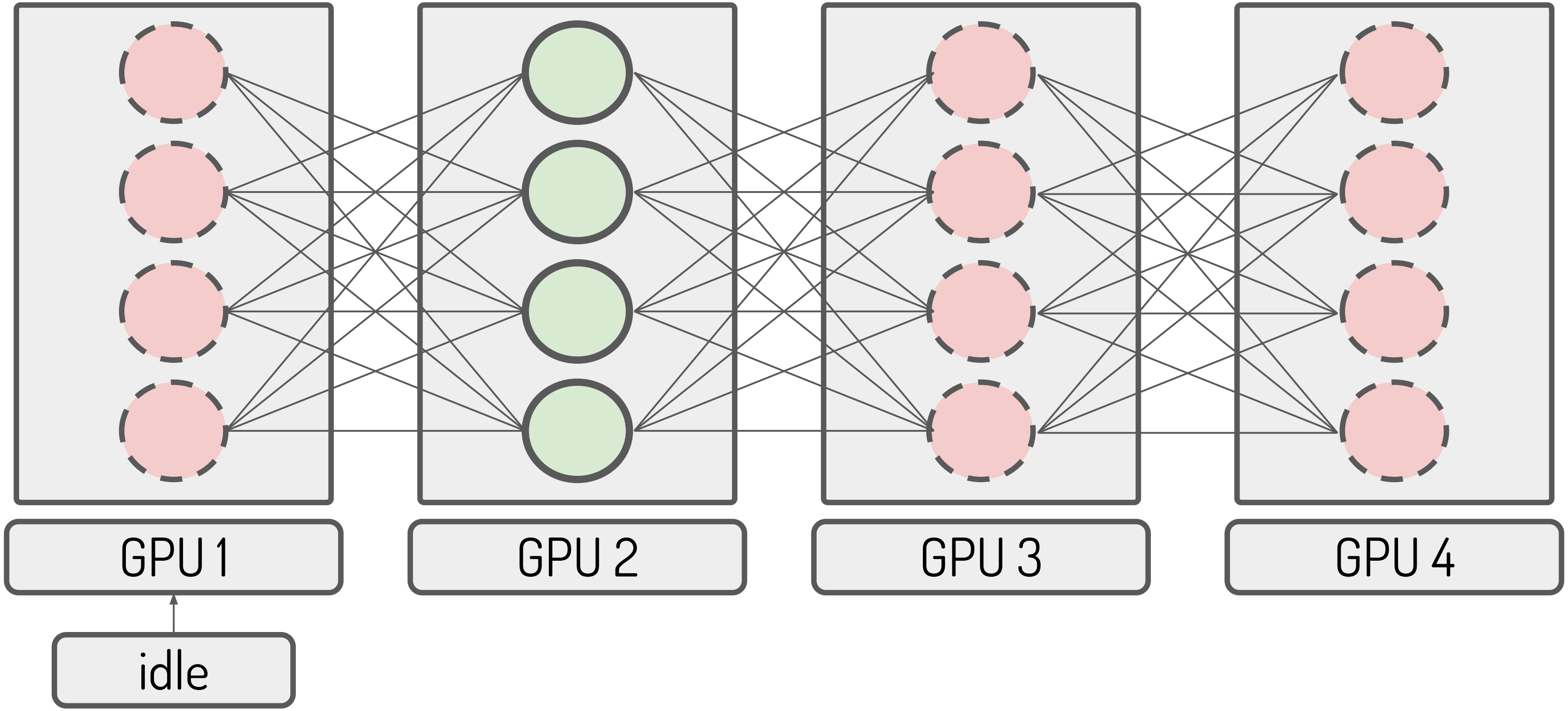
Model Parallelism: Naive



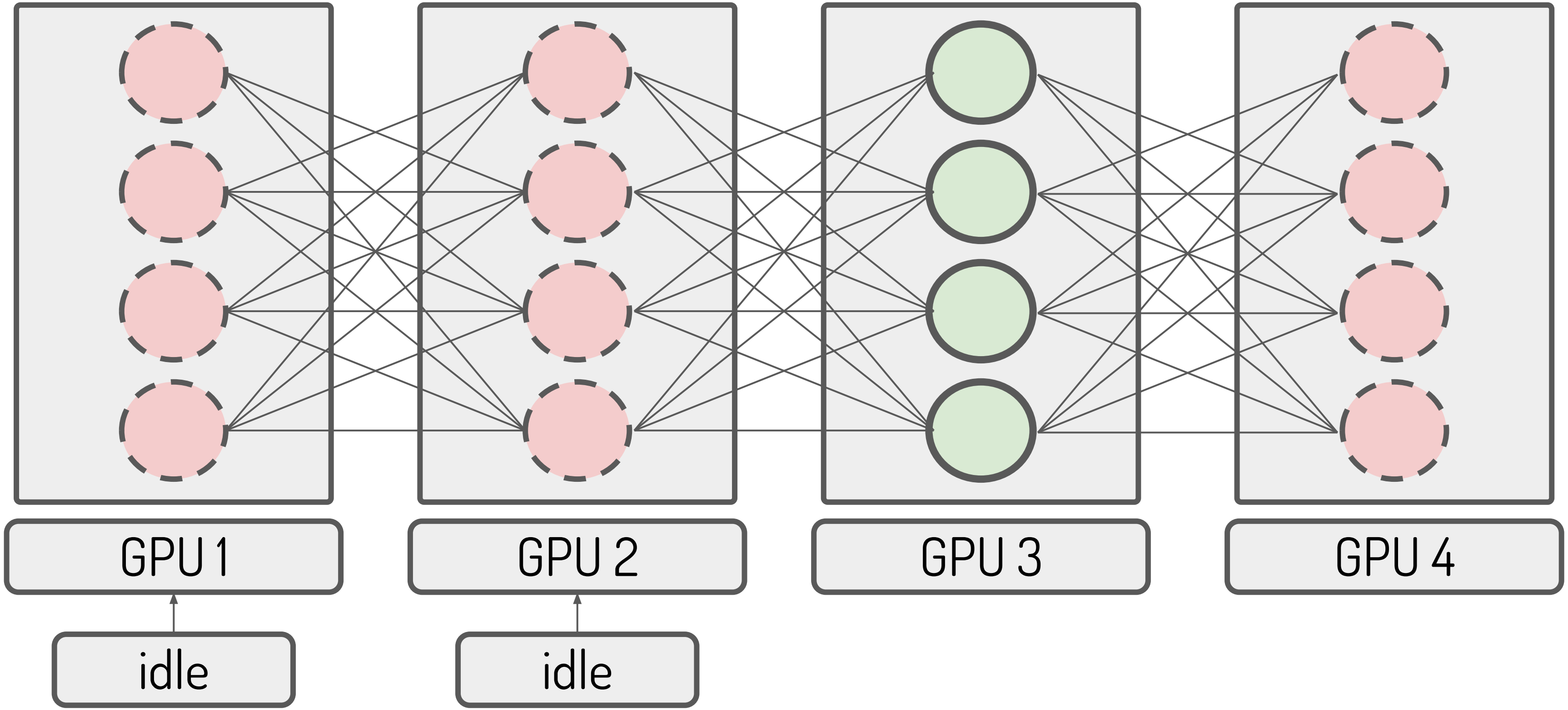
Model Parallelism: Naive



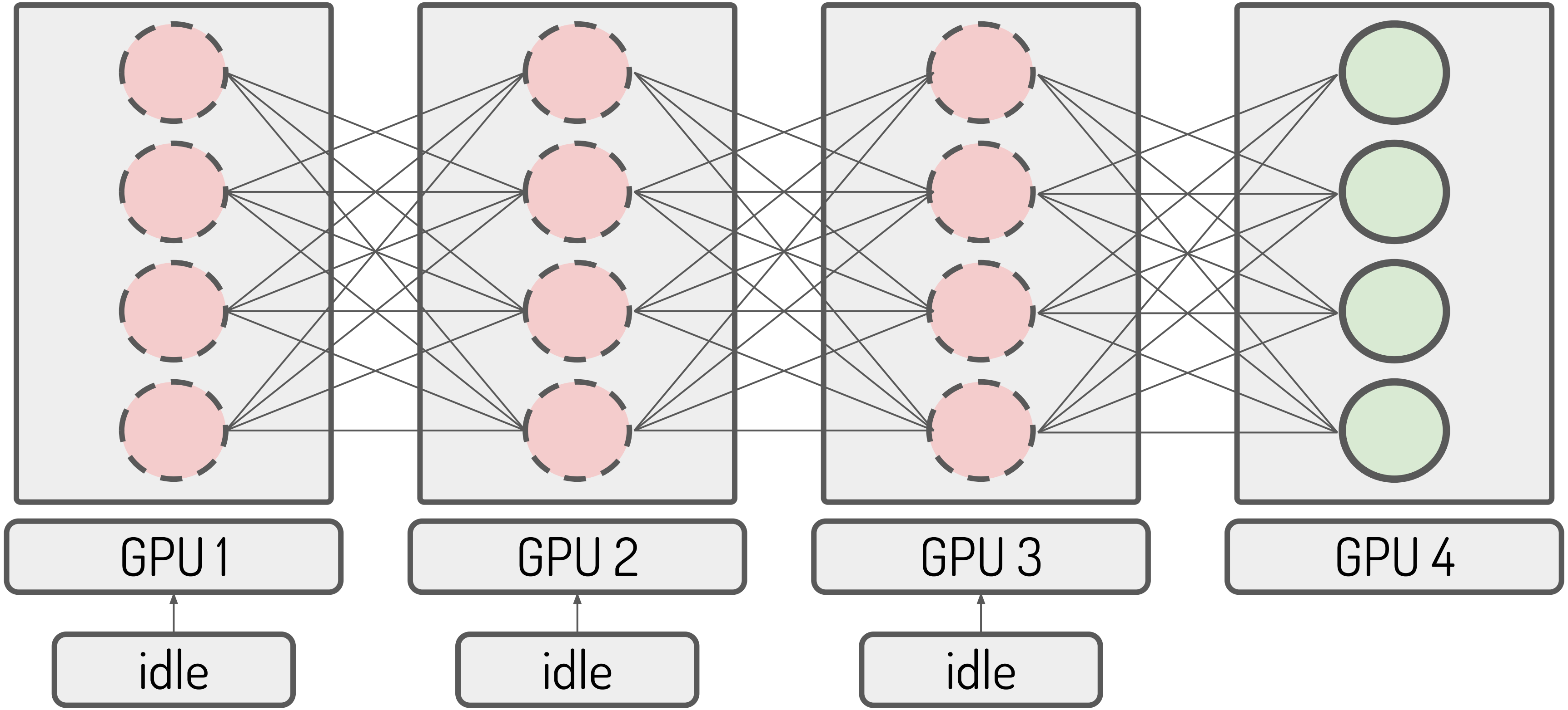
Model Parallelism: Naive



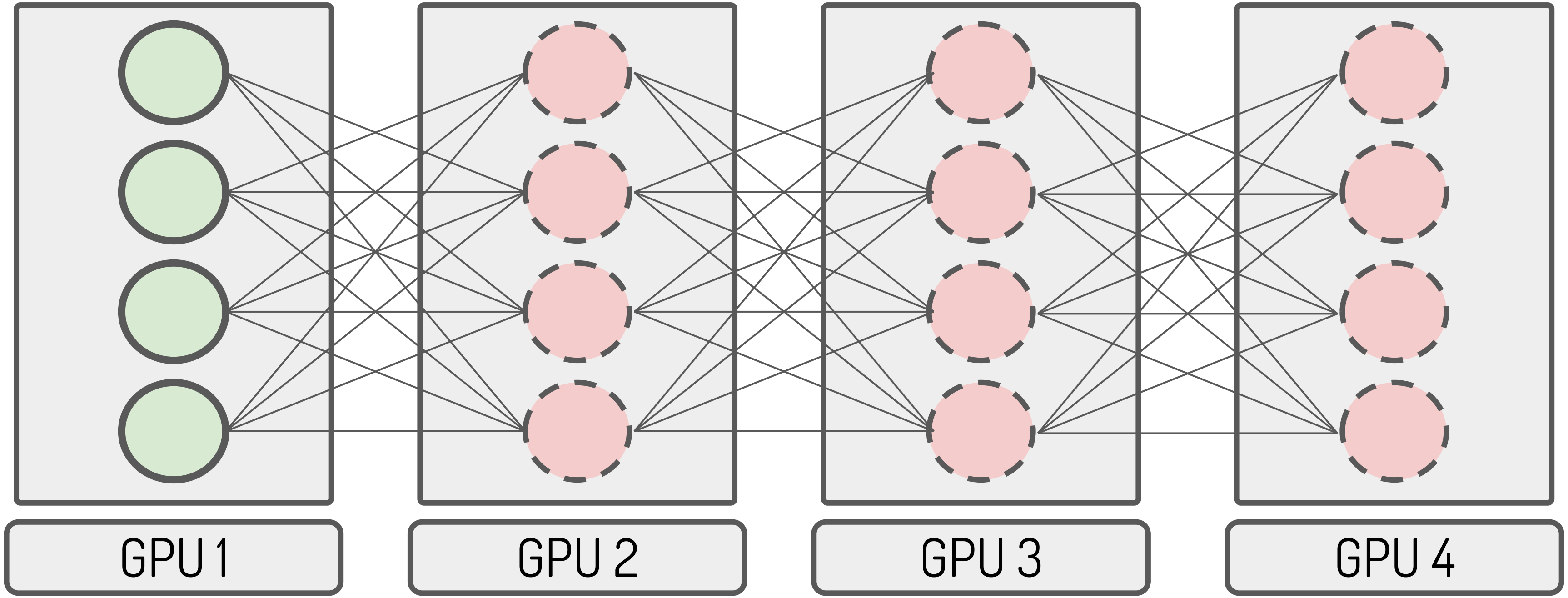
Model Parallelism: Naive



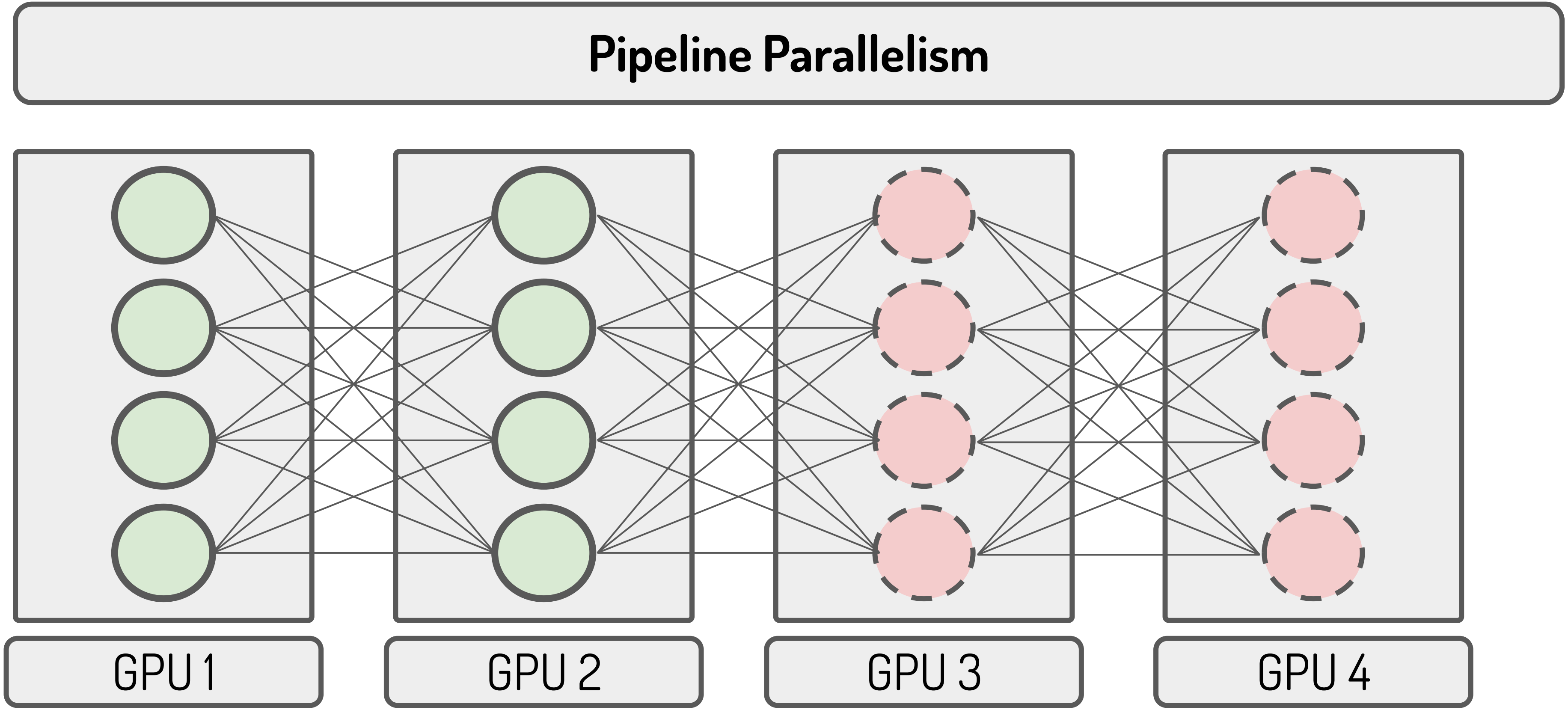
Model Parallelism: Naive



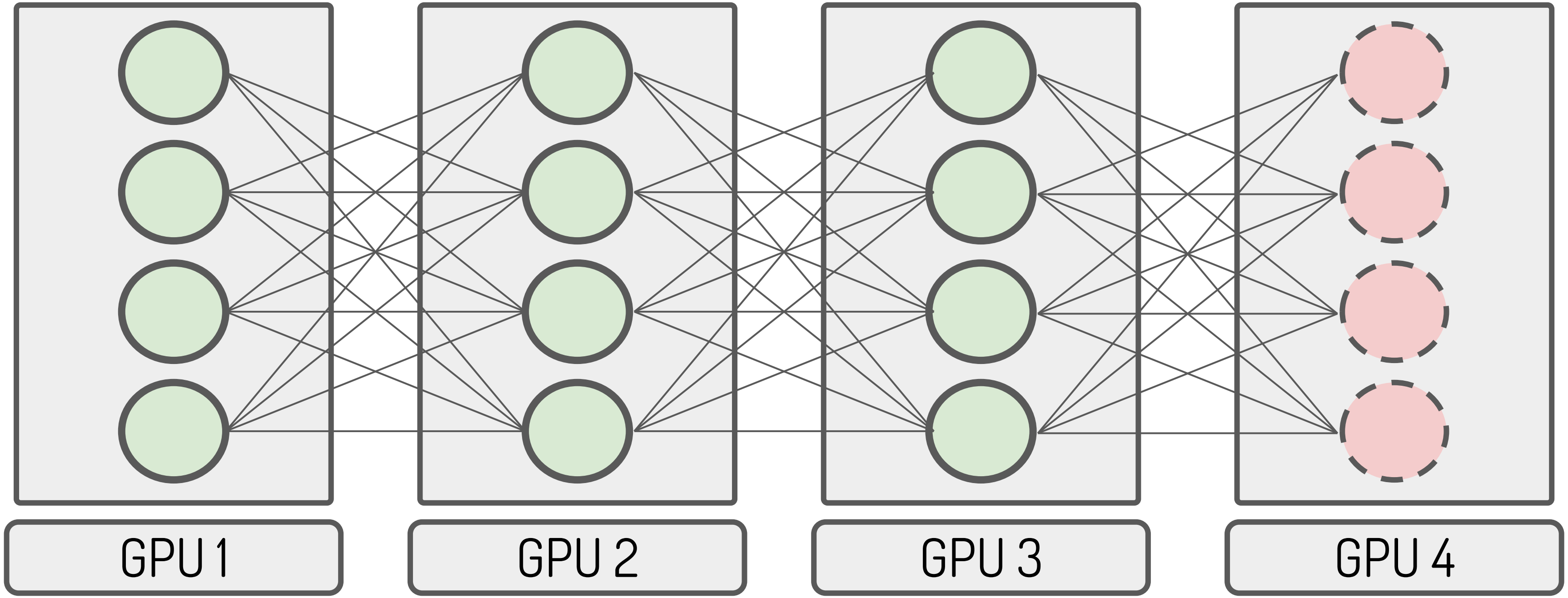
Pipeline Parallelism

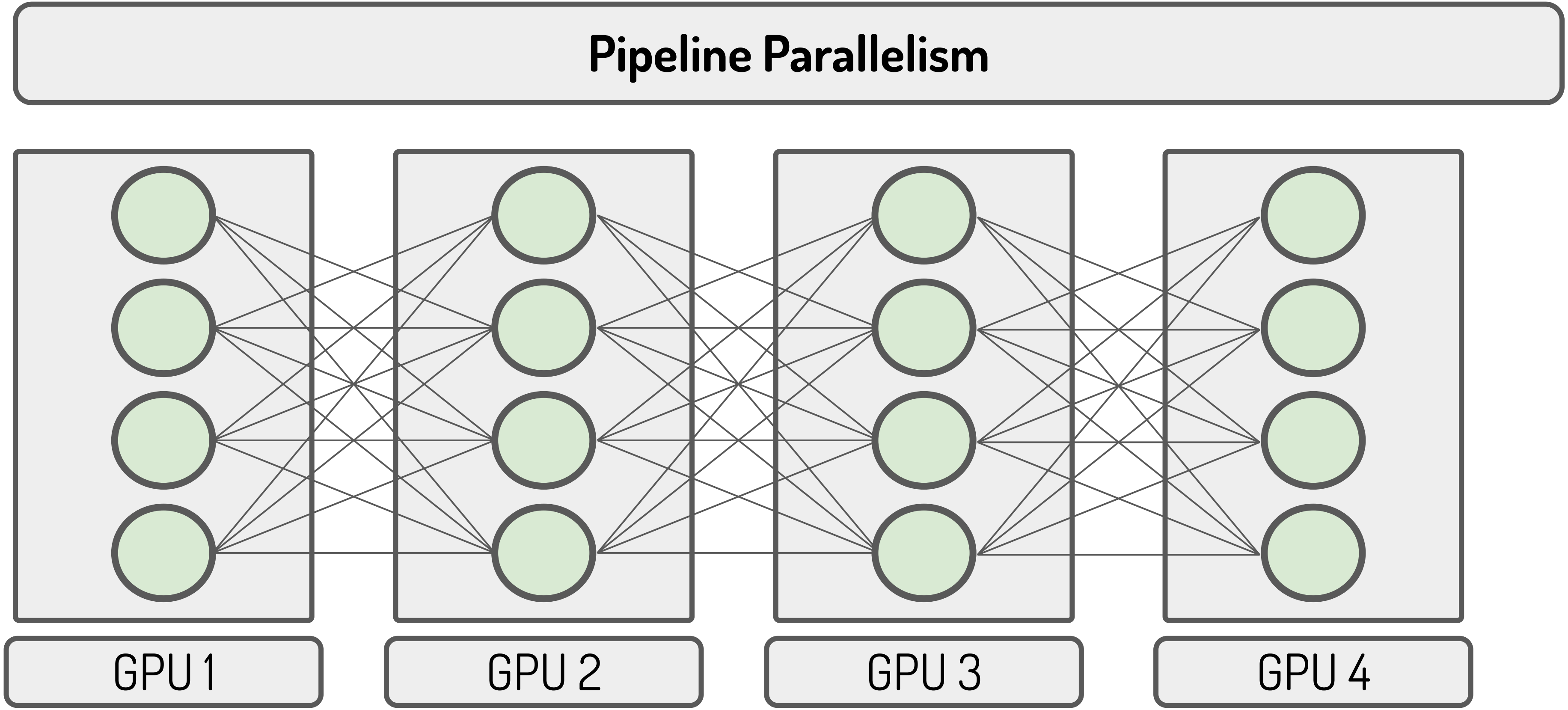


key idea: split mini-batch into sequential micro-batches


Pipeline Parallelism
GPU 1
GPU 2
GPU 3
GPU 4

Pipeline Parallelism




Pipeline Parallelism
GPU 1
GPU 2
GPU 3
GPU 4

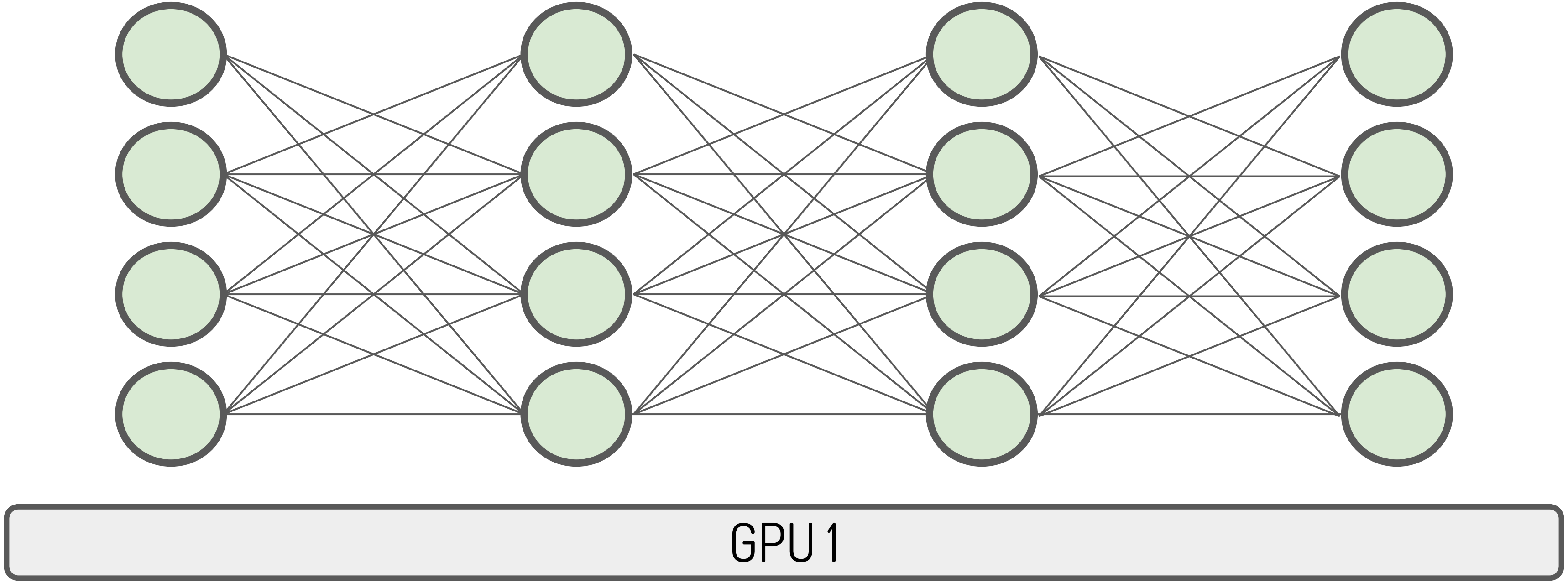
Storing Activations

Forward activations

major source of memory usage

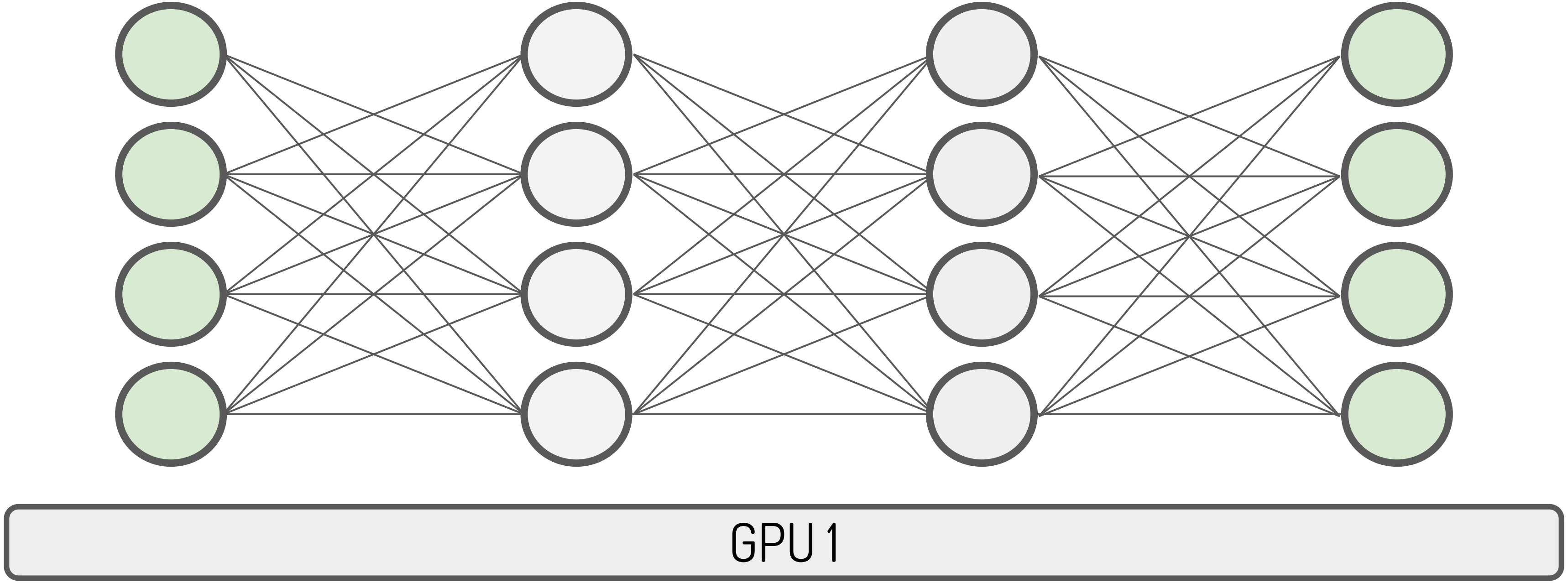
memory usage = minibatch size x # parameters

Solution 1: Gradient Checkpointing

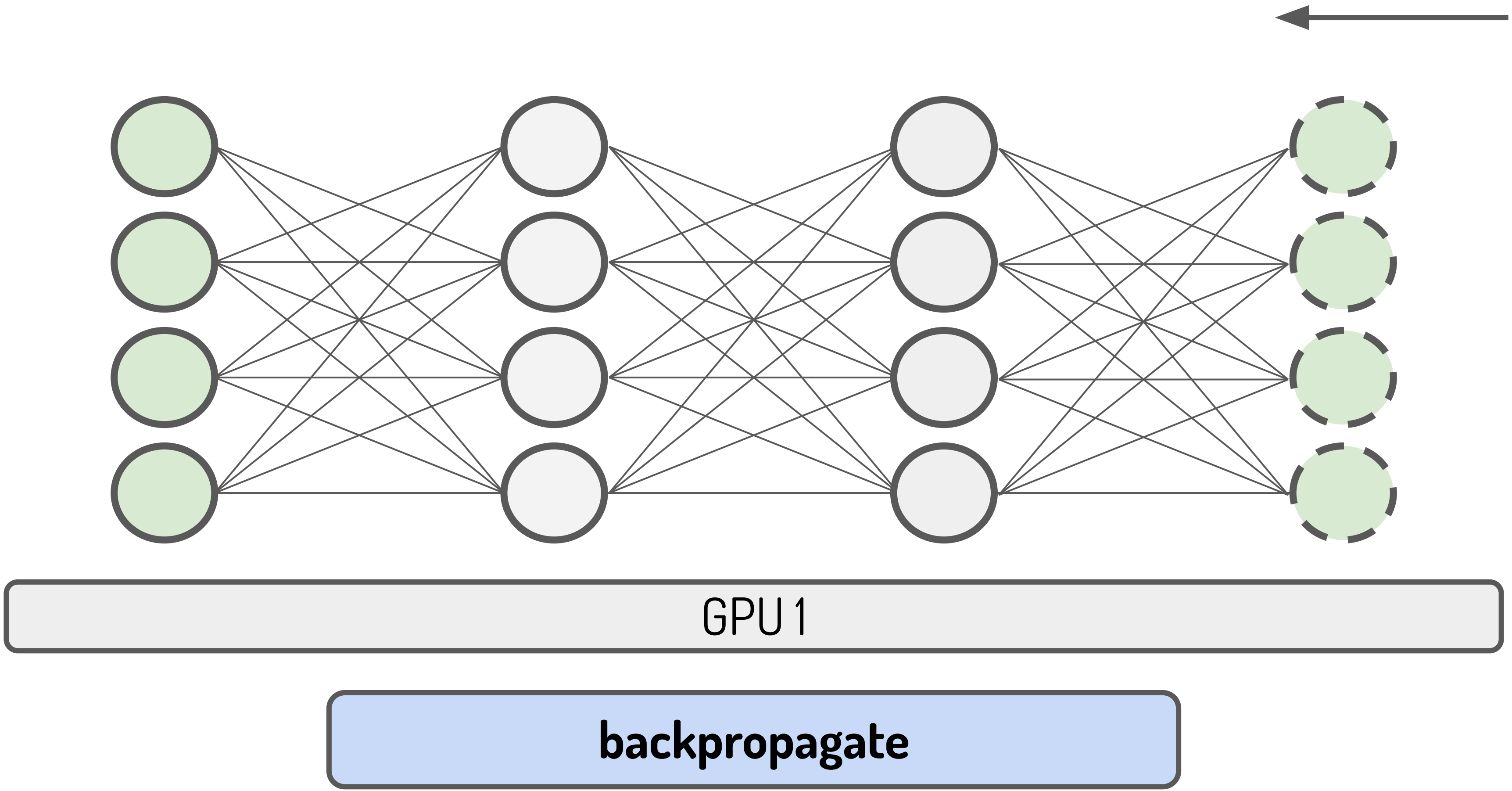


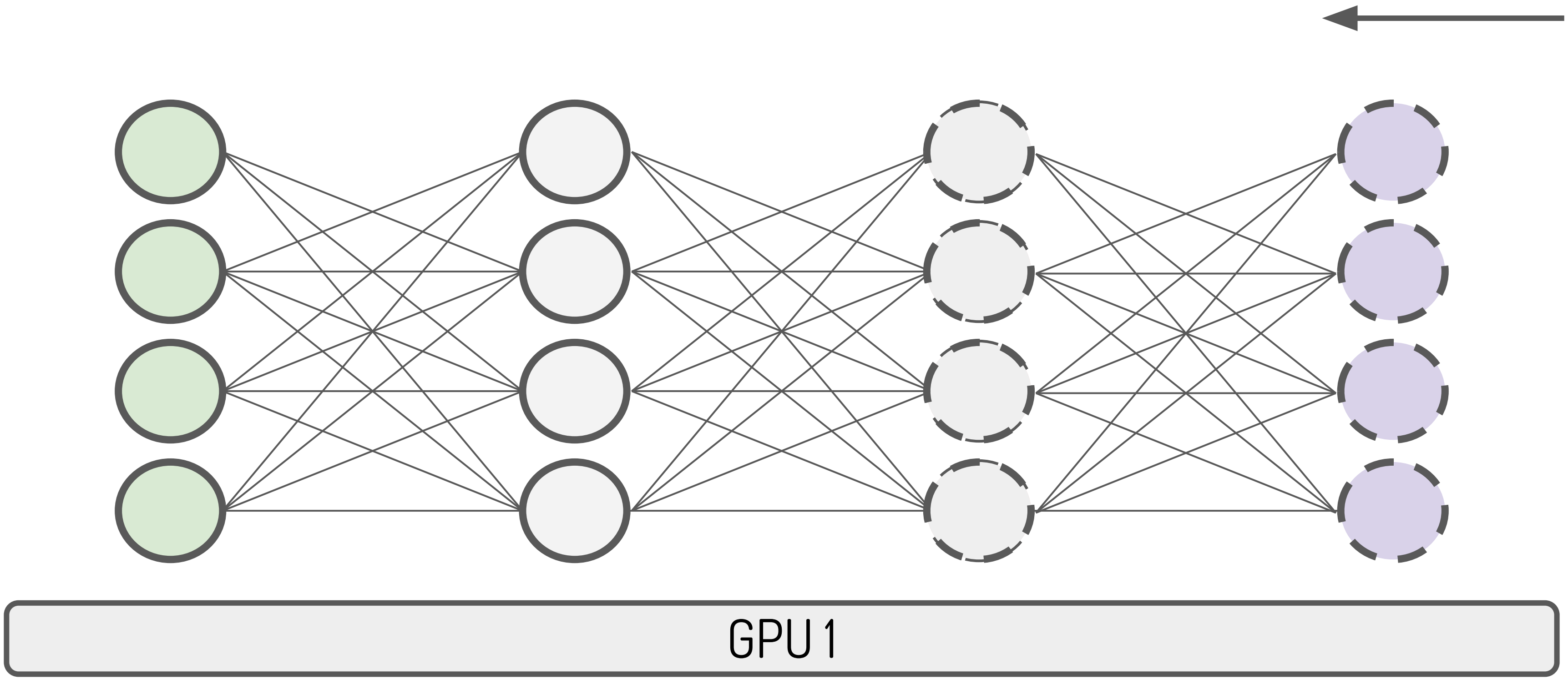
key idea: trade-off memory for compute

Solution 1: Gradient Checkpointing

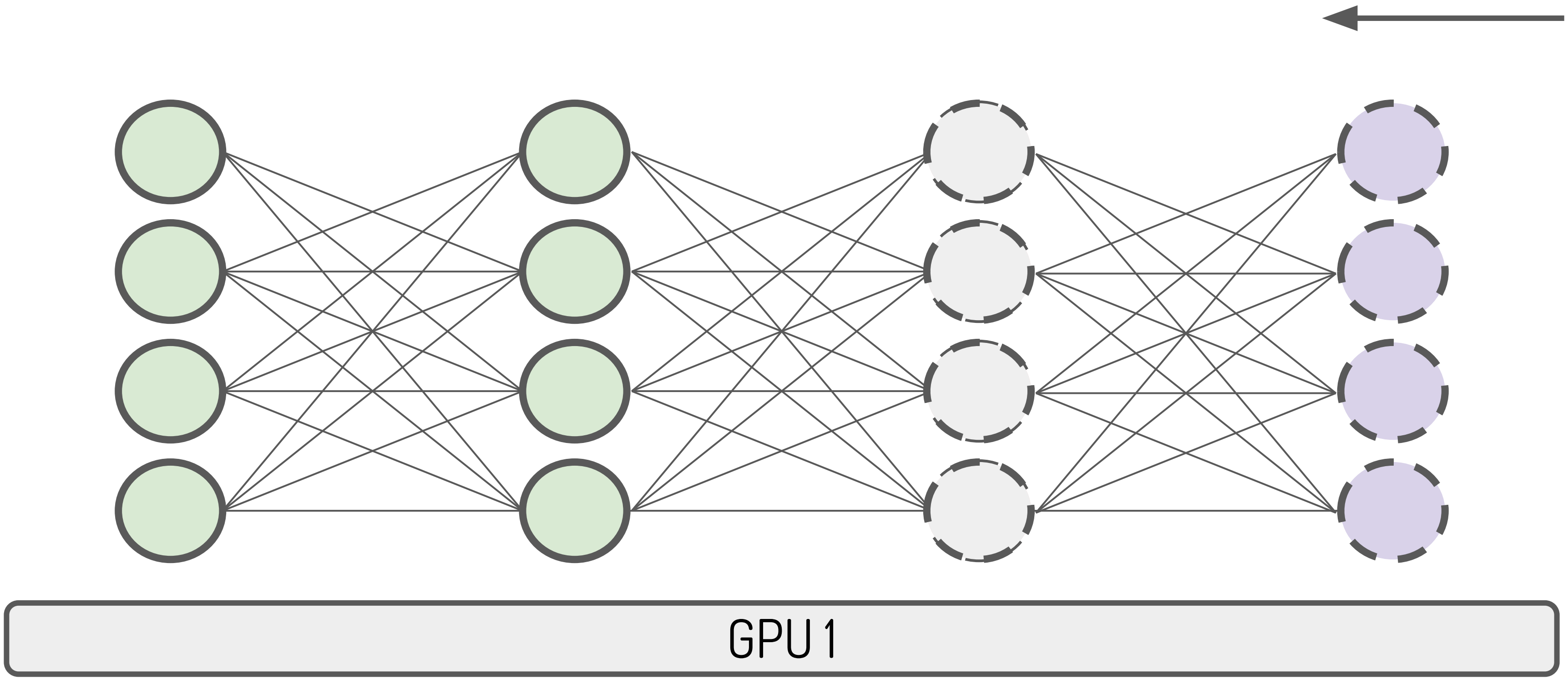


don’t store some activations in forward pass

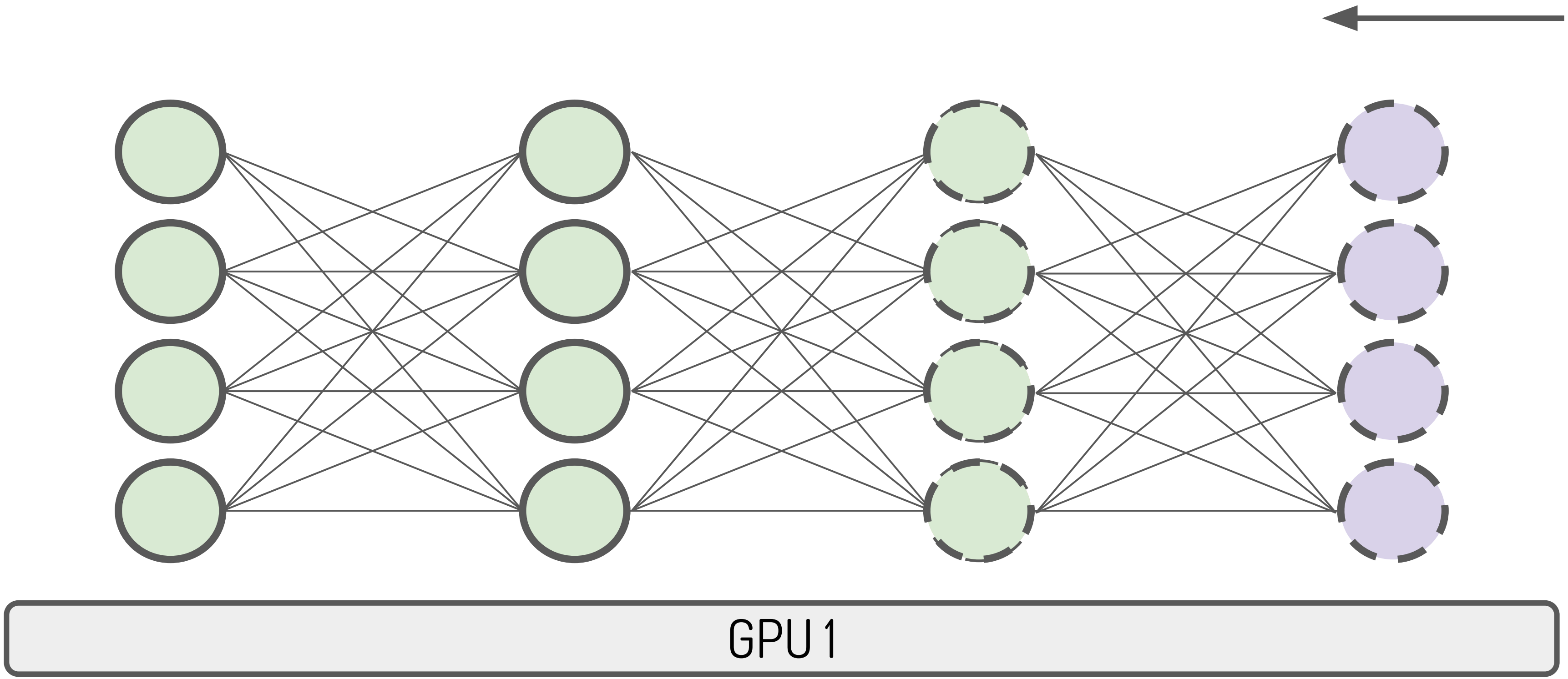

GPU 1
backpropagate



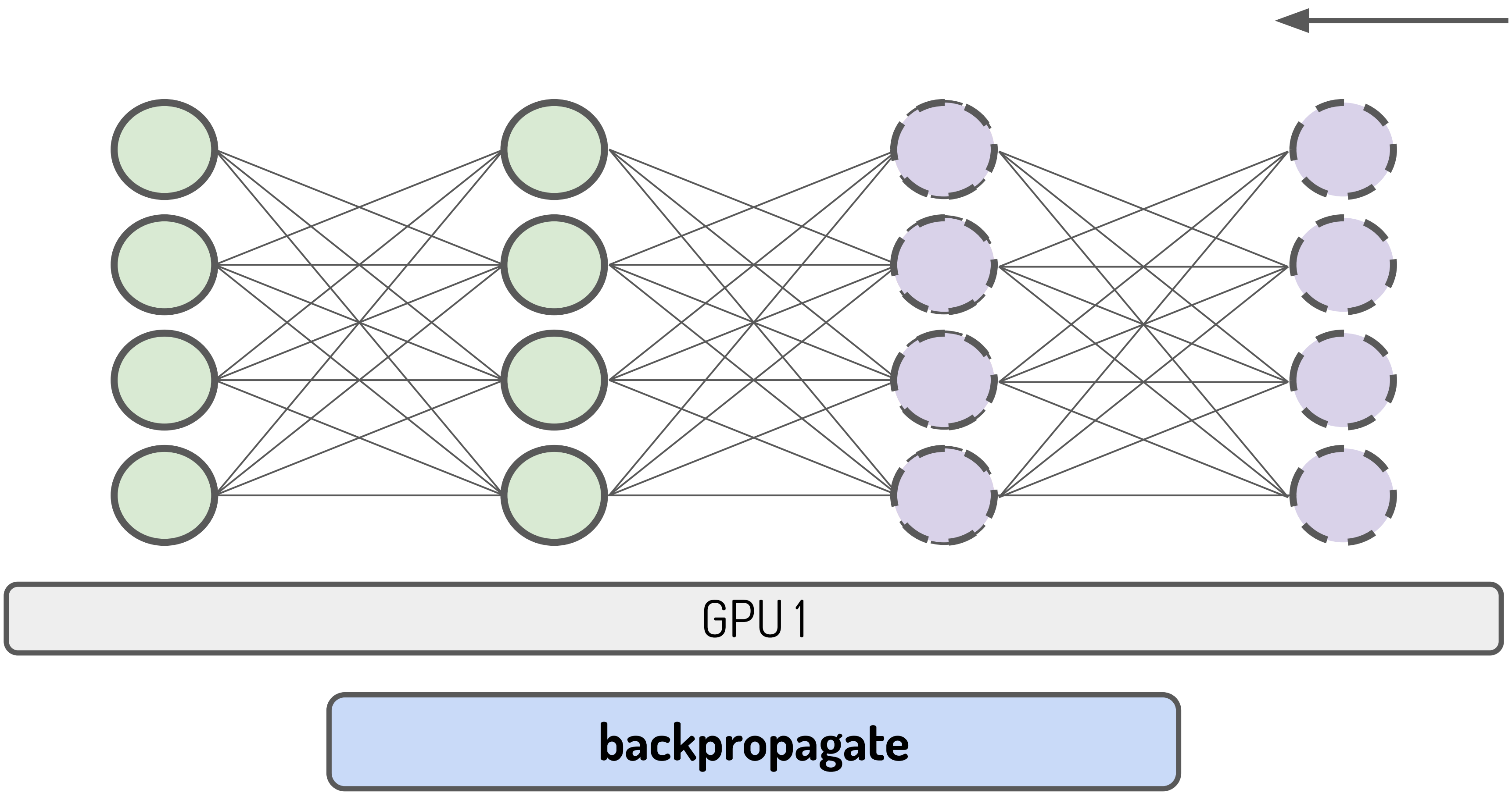
don’t have activations!

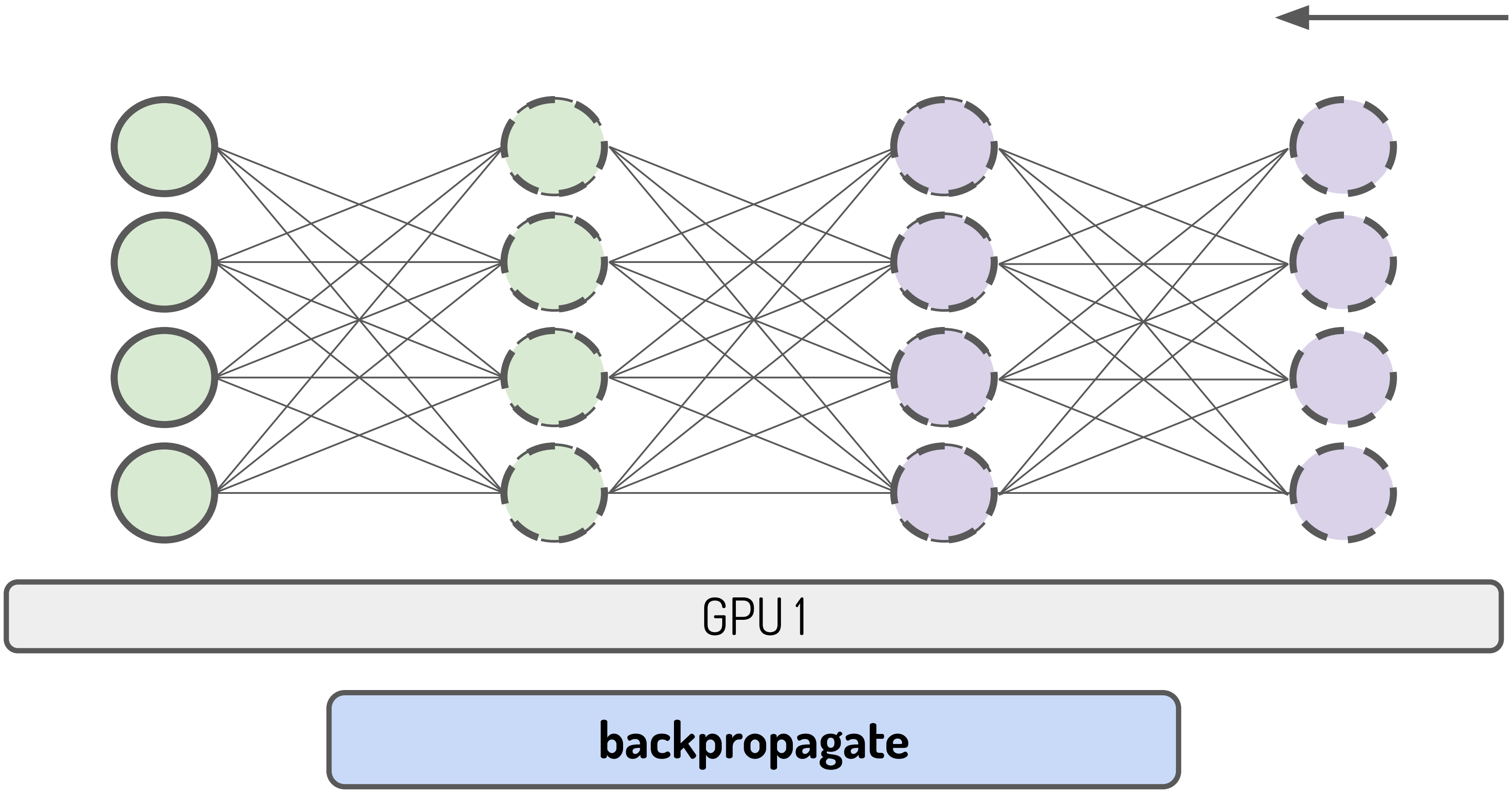


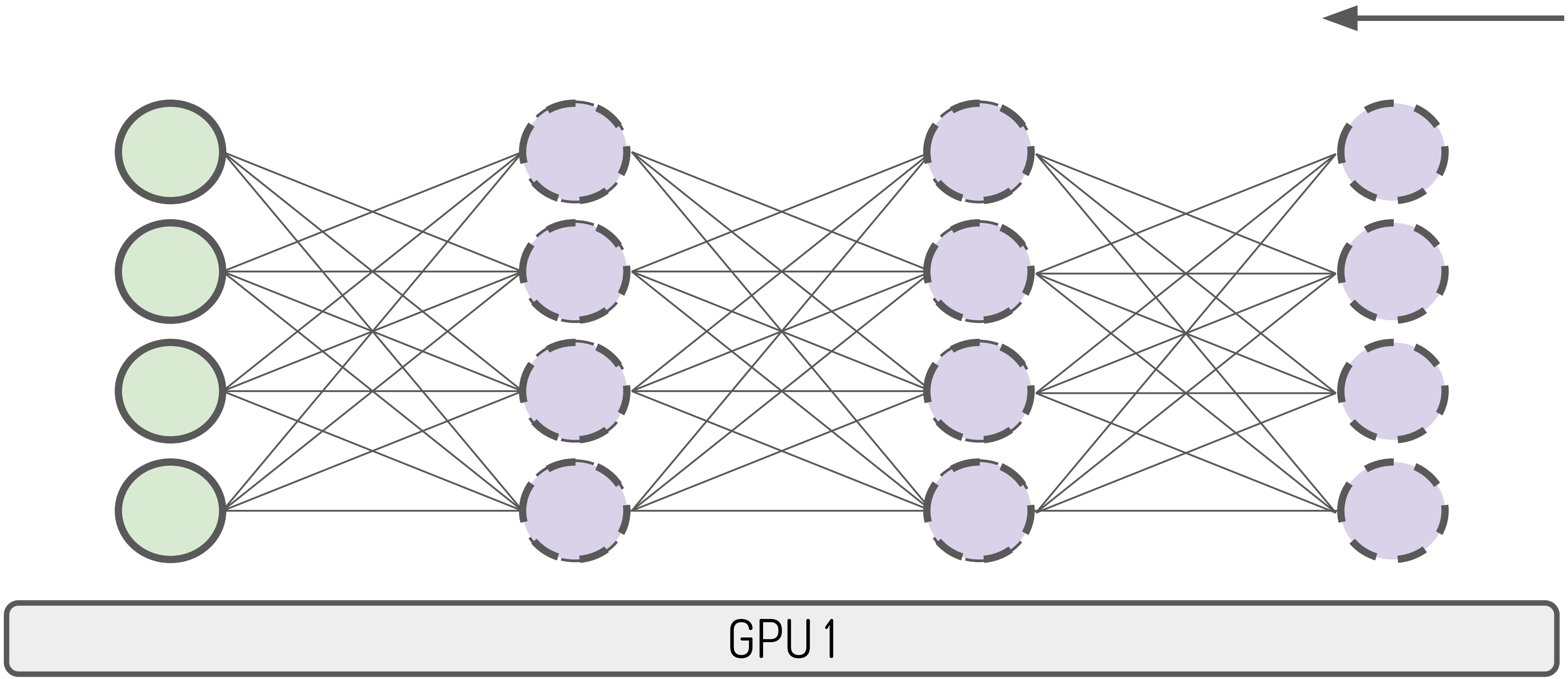
recompute activations from checkpoint



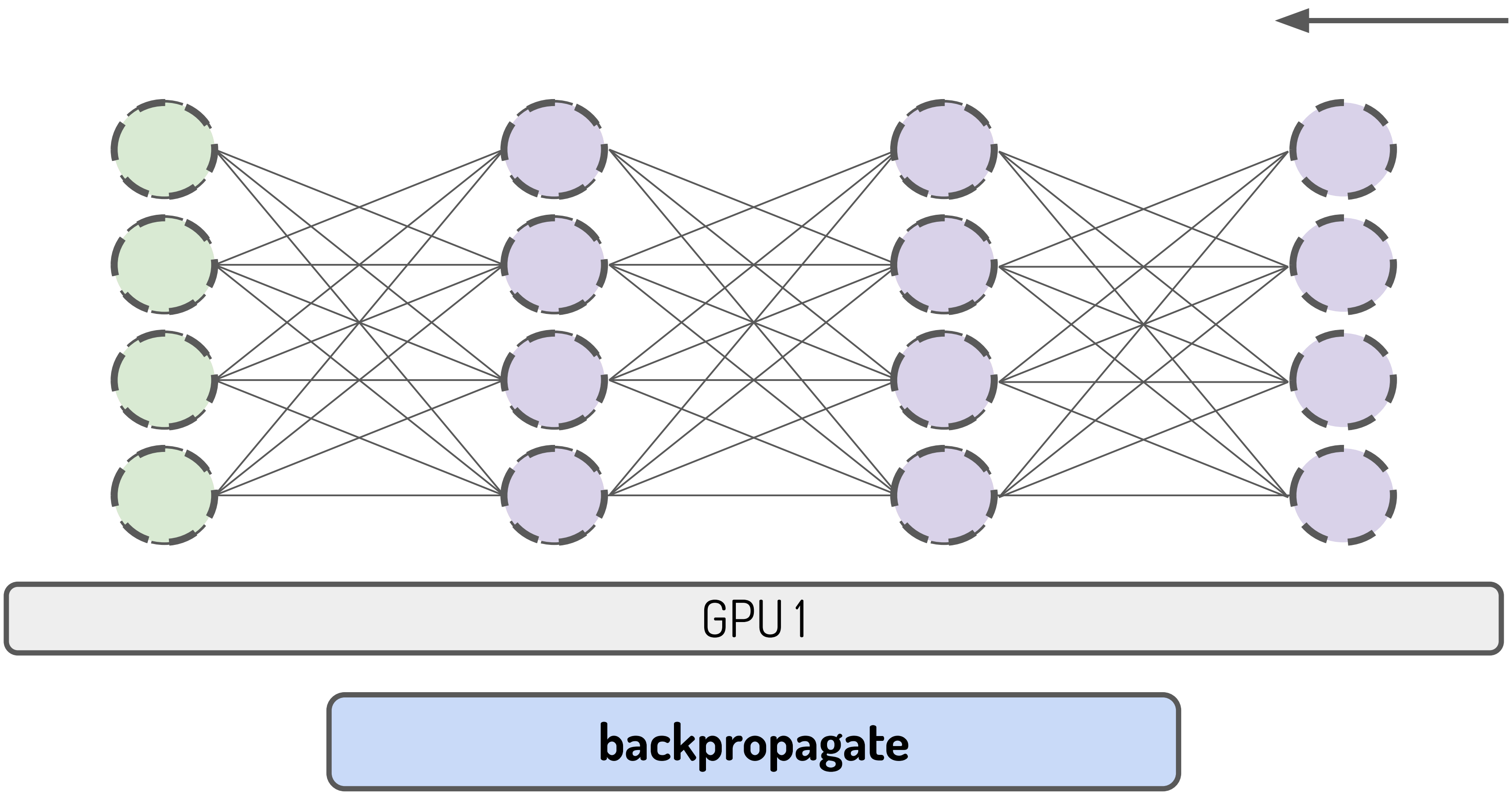
backpropagate

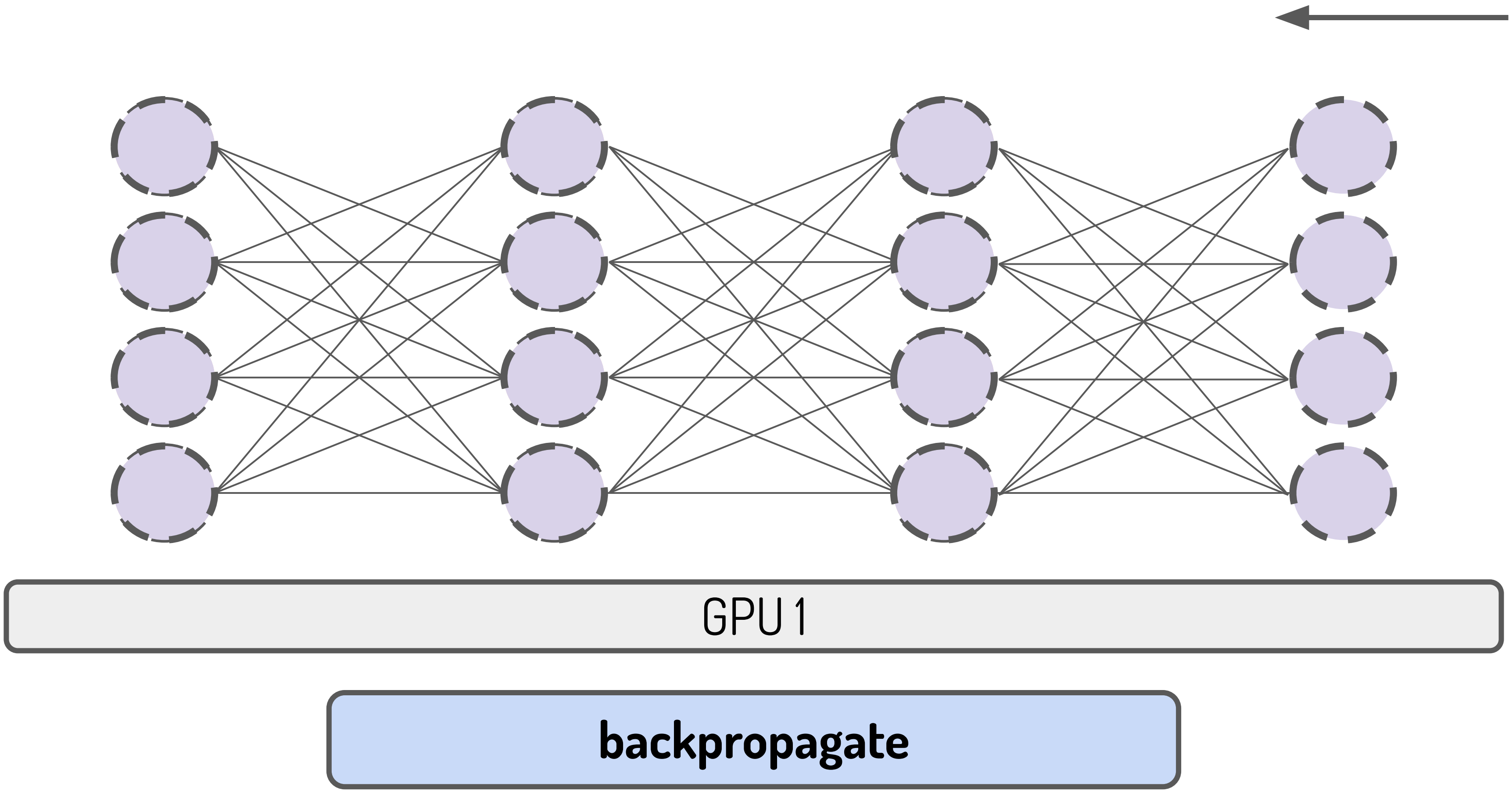

GPU 1
backpropagate


GPU 1
backpropagate



backpropagate


GPU 1
backpropagate


GPU 1
backpropagate

“For feed-forward models we were able to fit more than 10x larger models onto our GPU, at only a 20% increase in computation time.”

Credit: https://github.com/cybertronai/gradient-checkpointing

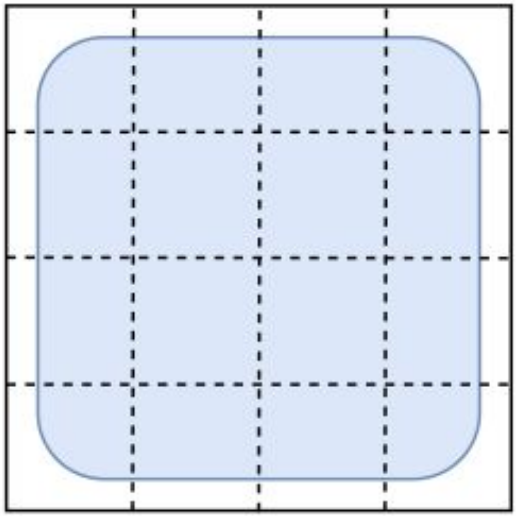
Solution 2: Data Parallelism for Large Batch Training

split the data across devices

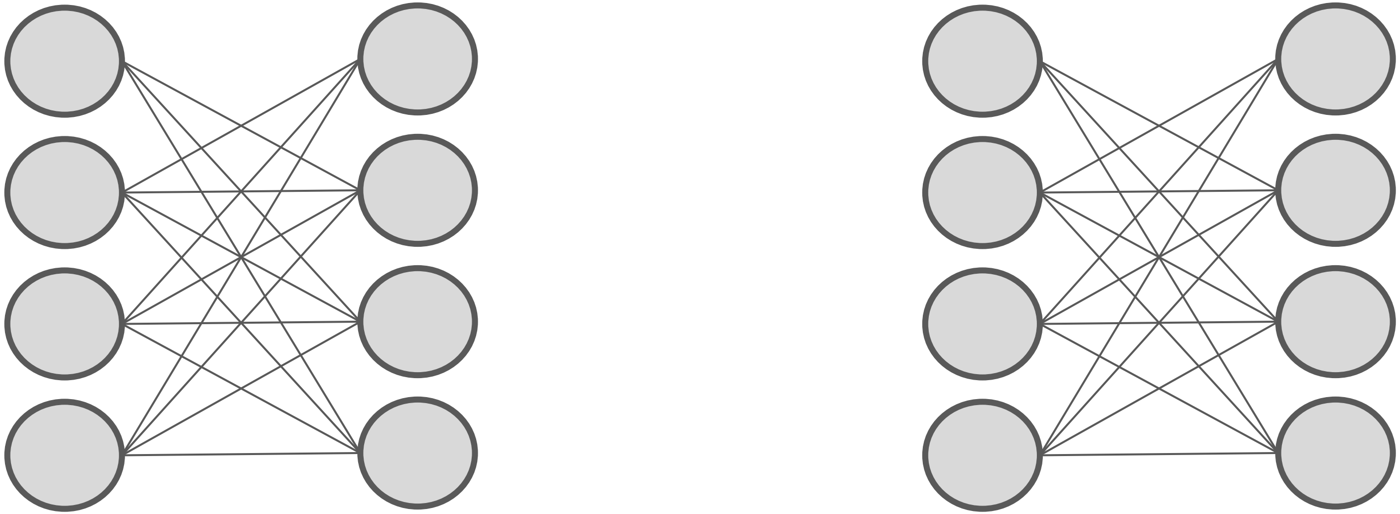
each device sees a fraction of the batch

each device replicates the model

each device replicates the optimizer



replicate model across devices

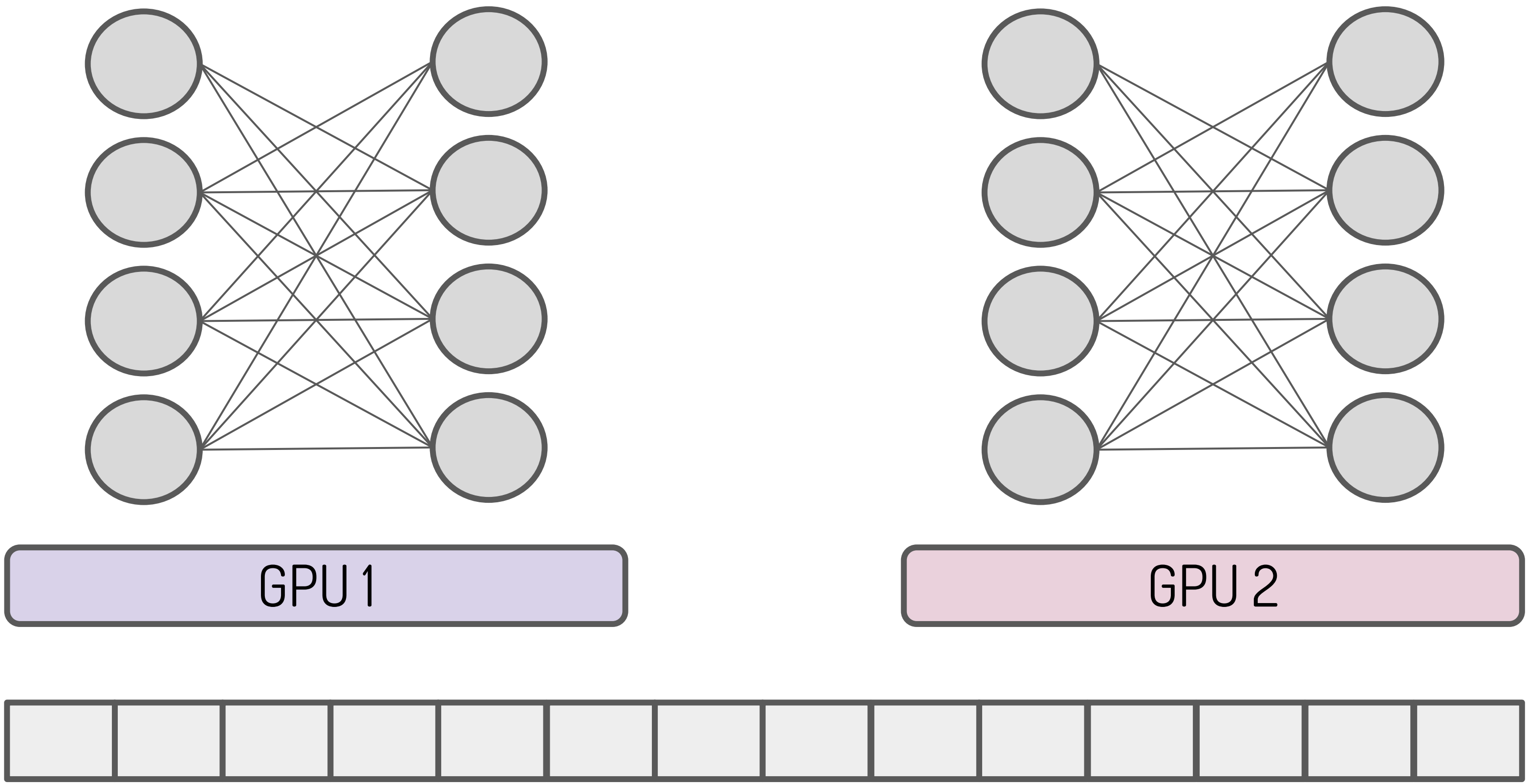


GPU 1

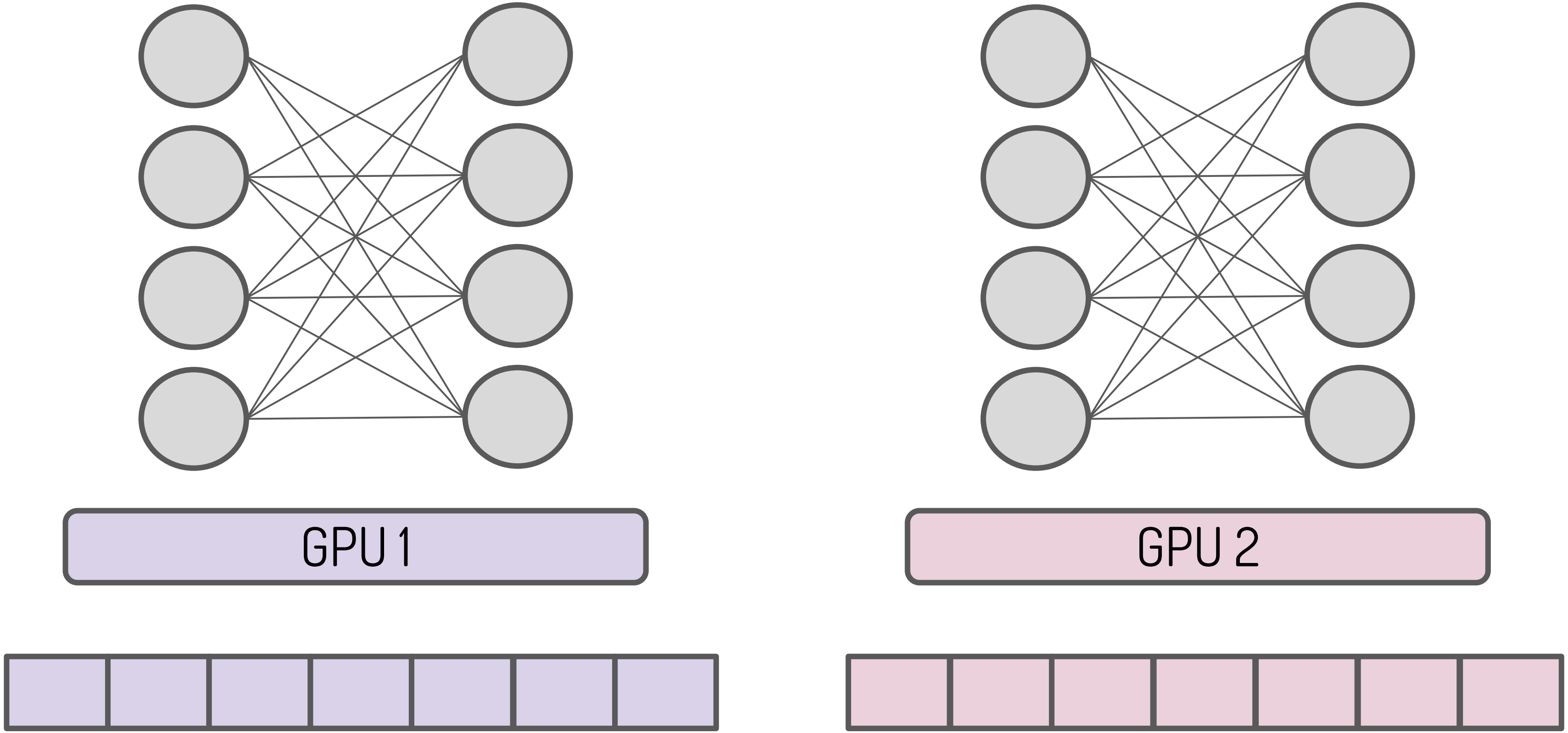
GPU 2

GPUs could be on same or multiple nodes

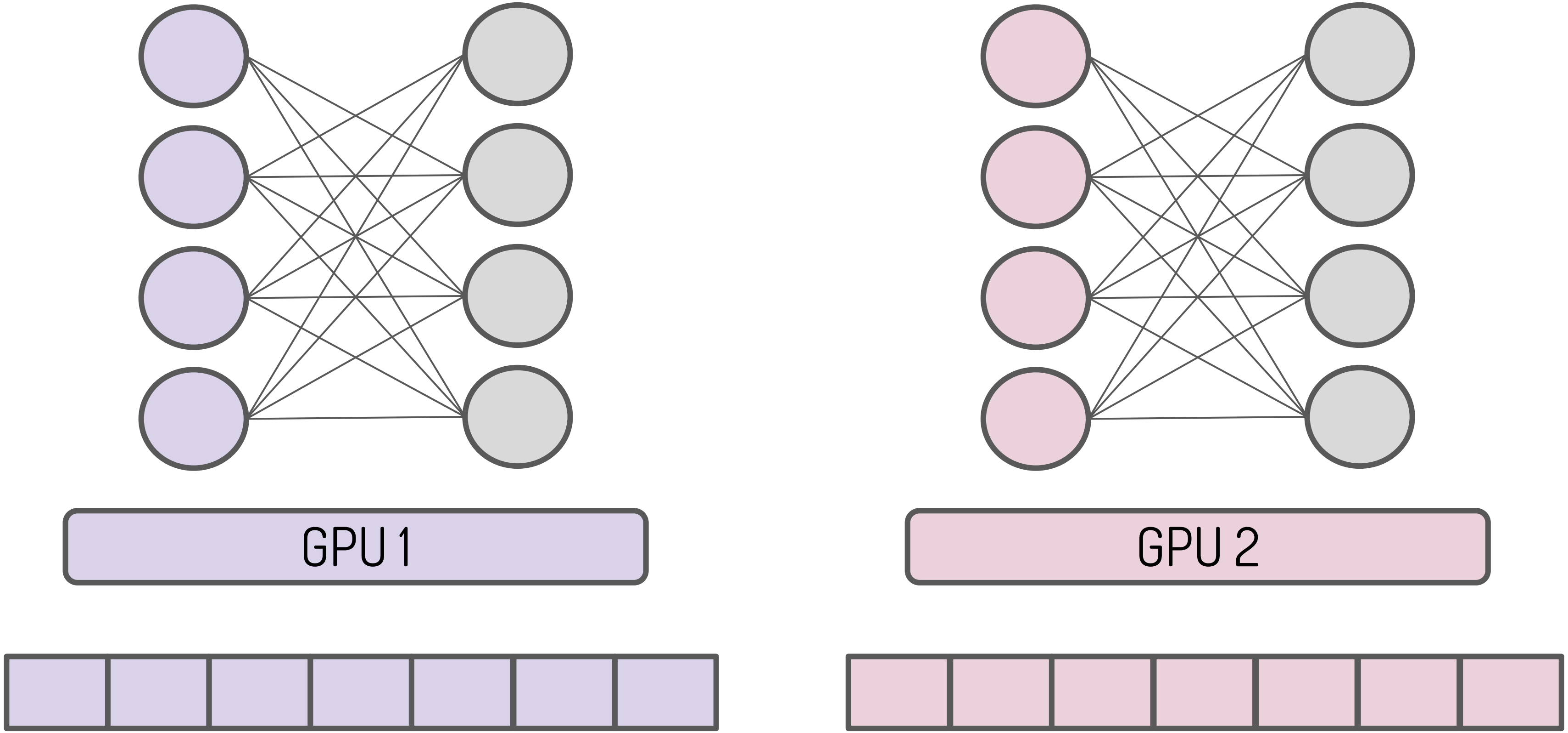
to push in a batch of data



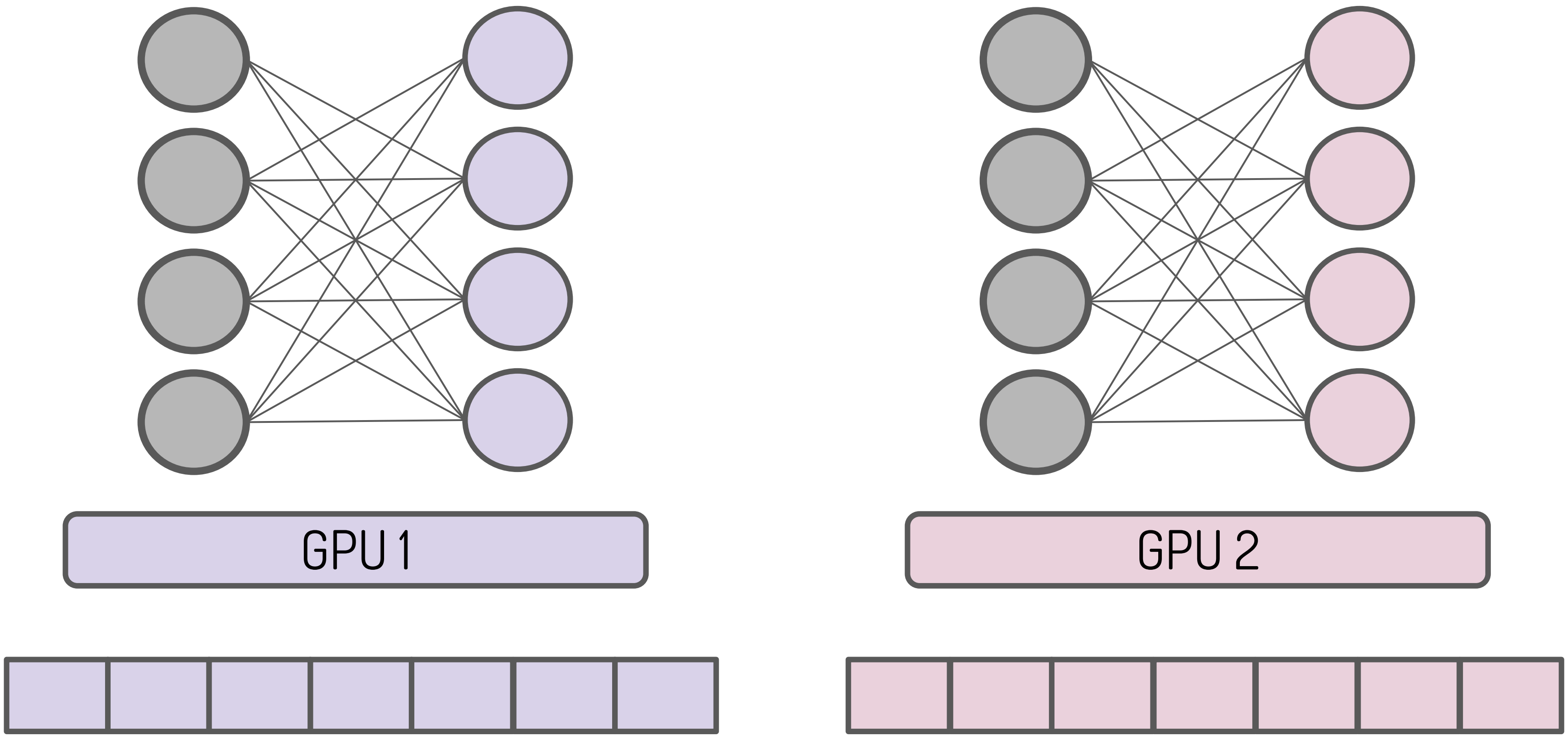
split batch across devices

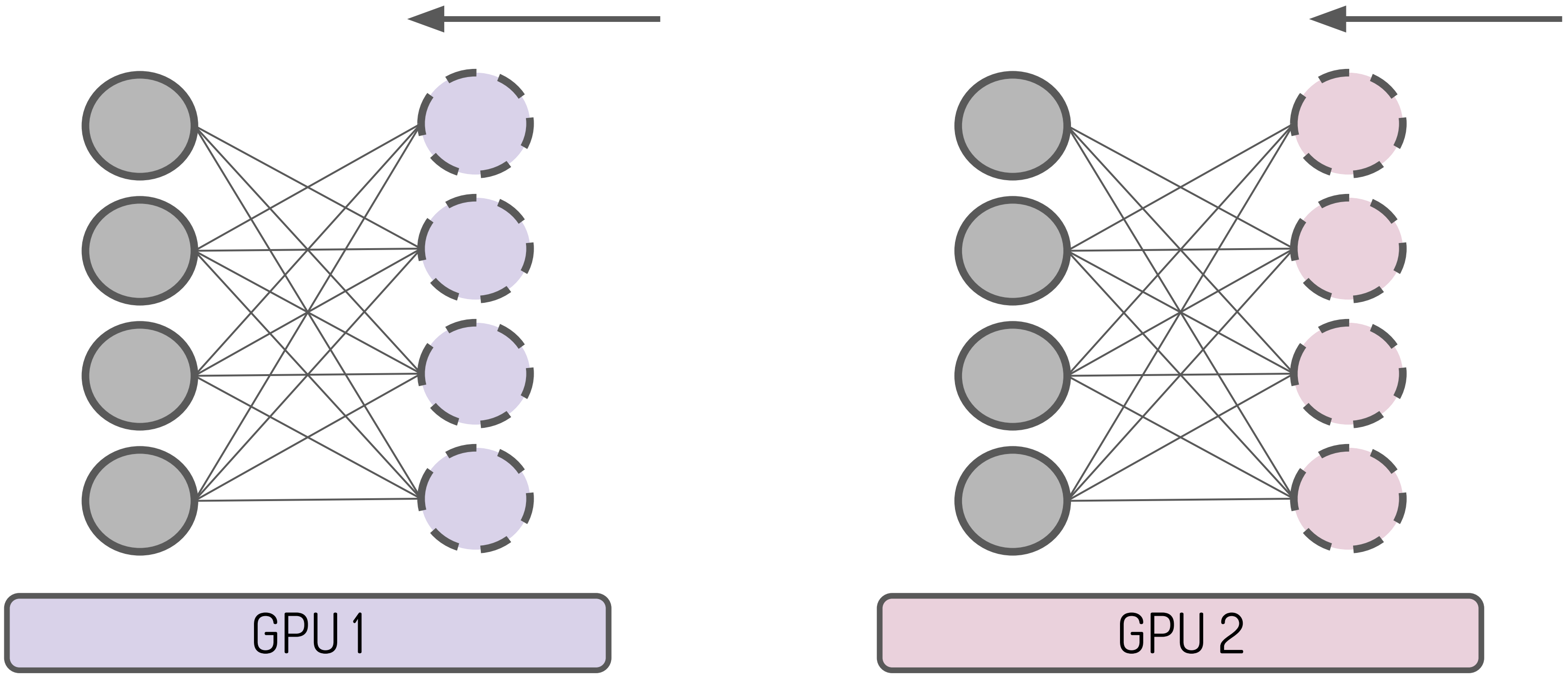


parallel forward passes

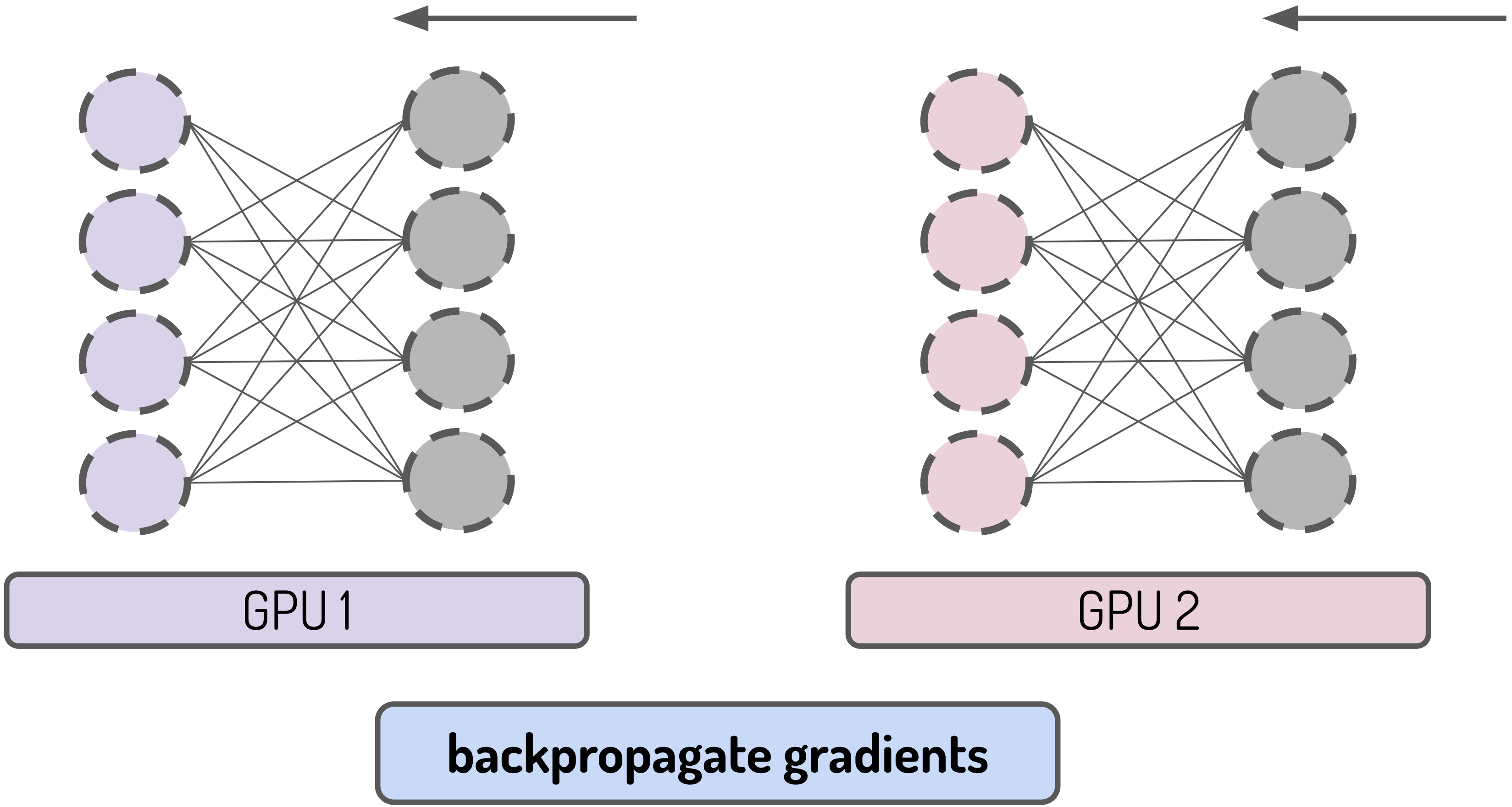


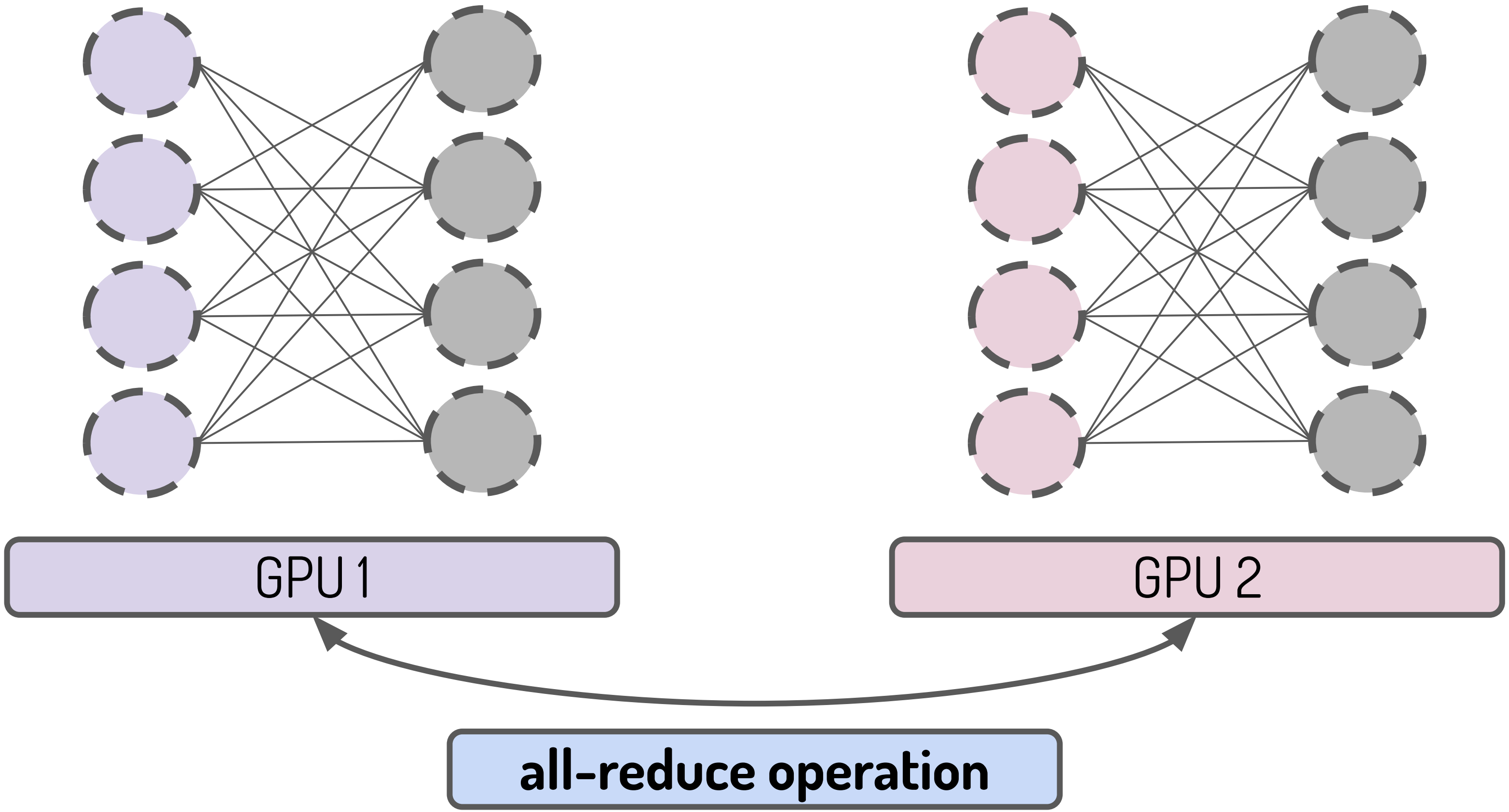
parallel forward passes

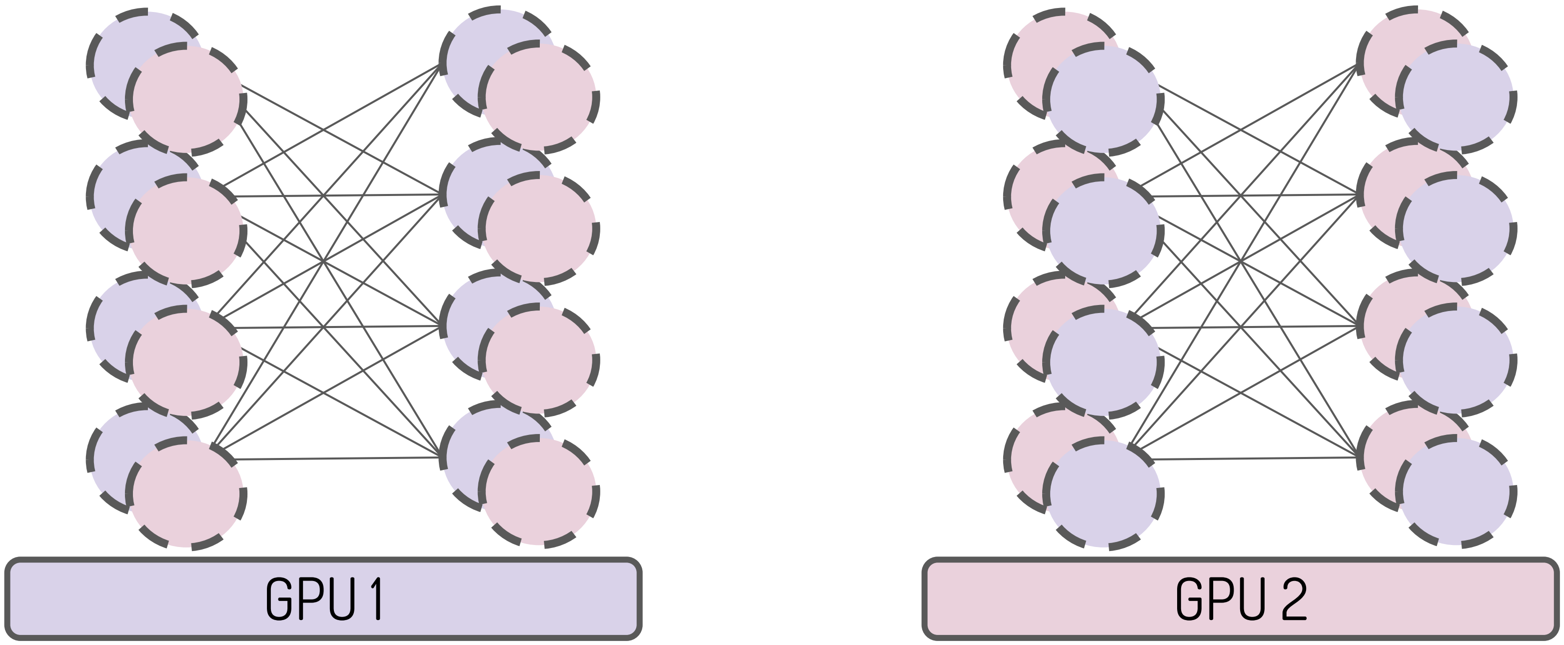




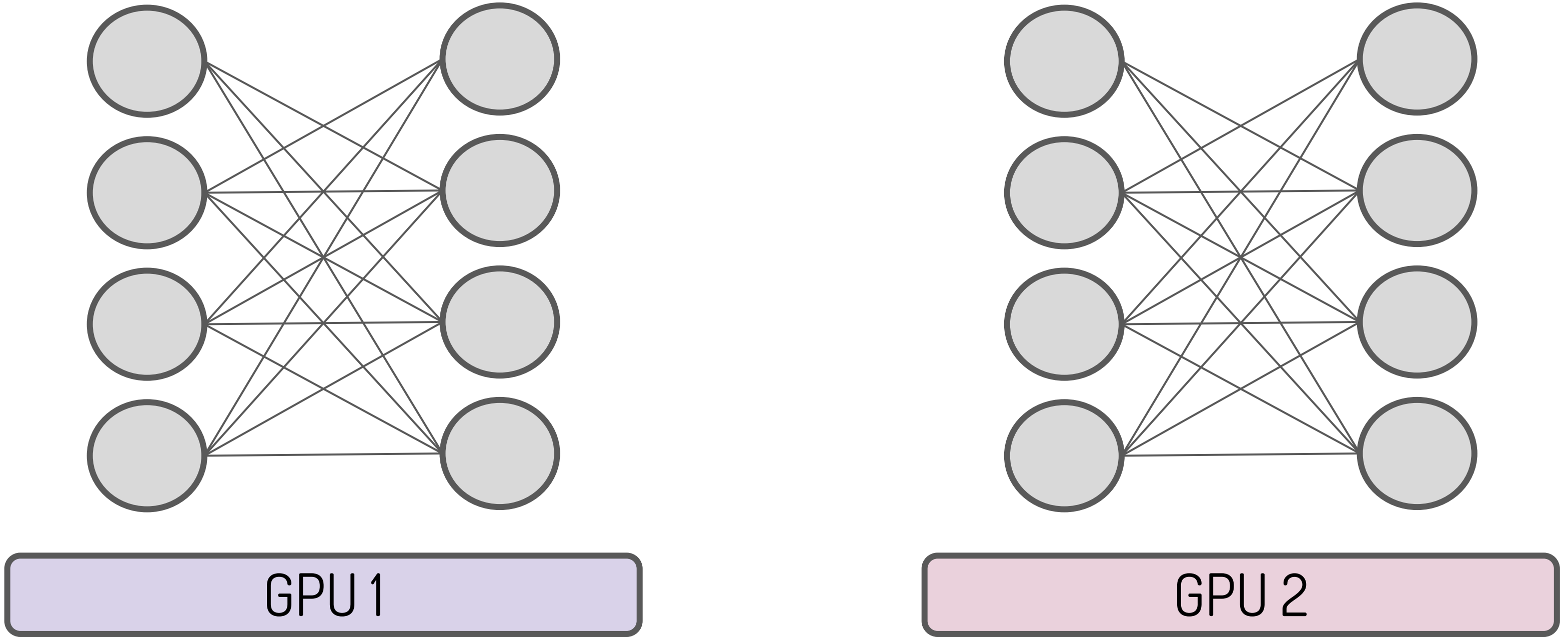
backpropagate gradients


GPU 1
GPU 2
backpropagate gradients


GPU 1
GPU 2
all-reduce operation



all devices do the same gradient updates



all parameters stay synchronized!

Trick: Gradient Bucketing

interleave communication with computation

synchronize buckets of gradients

Collective Communication

single- and multi-node communication

Collective Communication

single- and multi-node communication

Message Passing Interface (MPI)
Sets standard + CPU-CPU communication

synchronization, data movement, reduction

nVidia Collective Communications Library (nccl)
Follows MPI standard for GPU-GPU communication

Facebook Gloo
Optimized for ML: CPU-CPU/GPU-GPU communication

Inter-Process Communication: The All-Reduce

all-reduce operation

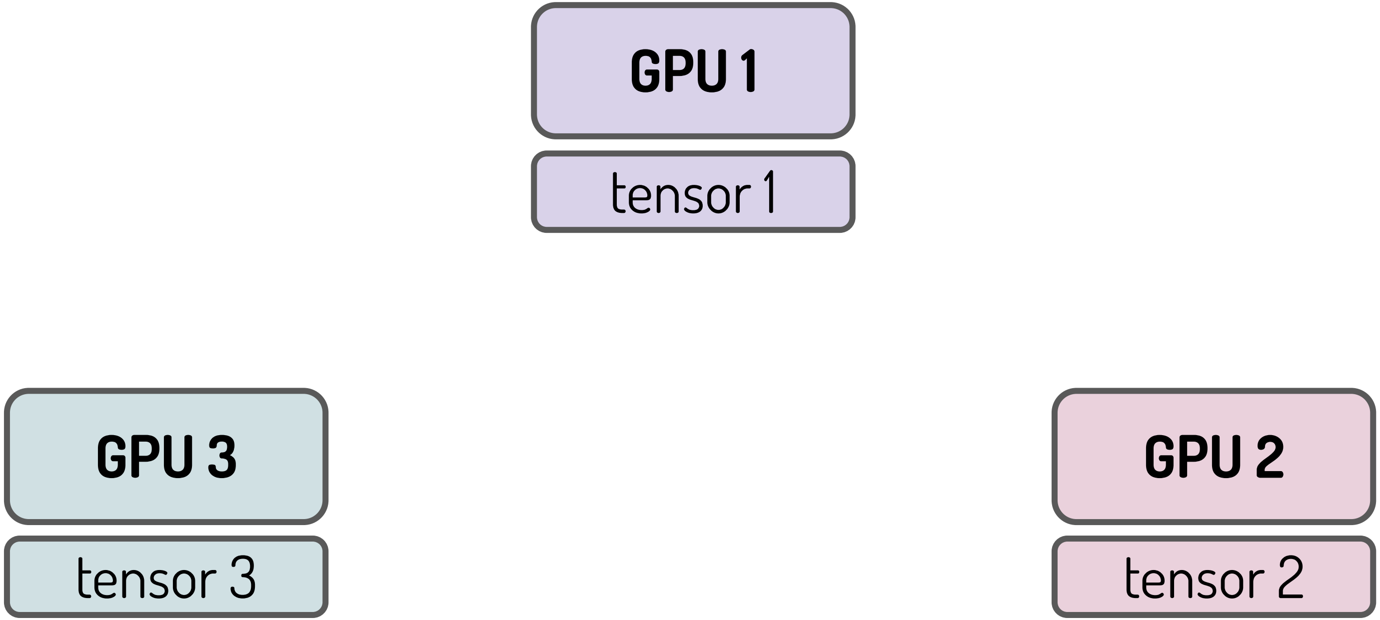
p processes

each process has tensor of size n

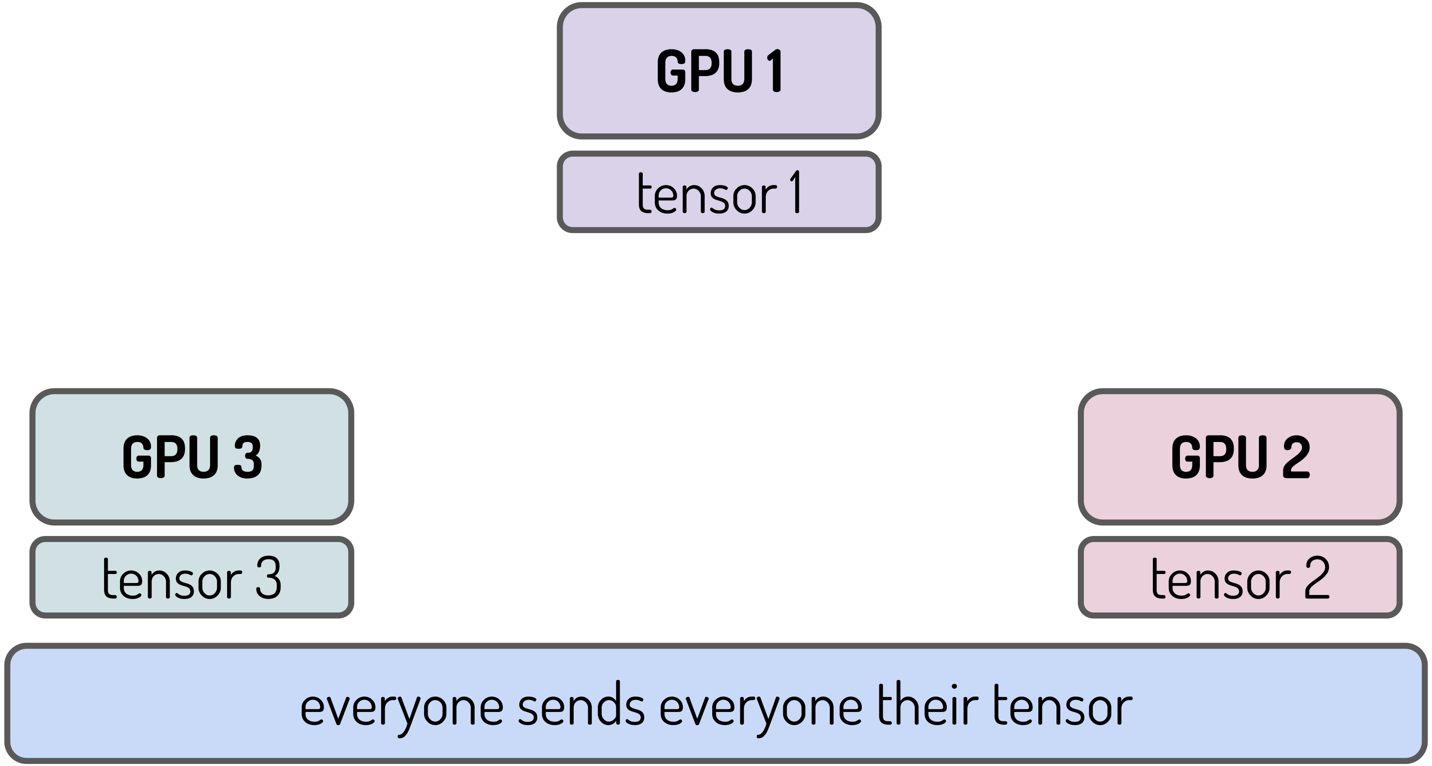
tensors aggregated (e.g. *sum*)

result returned to each process

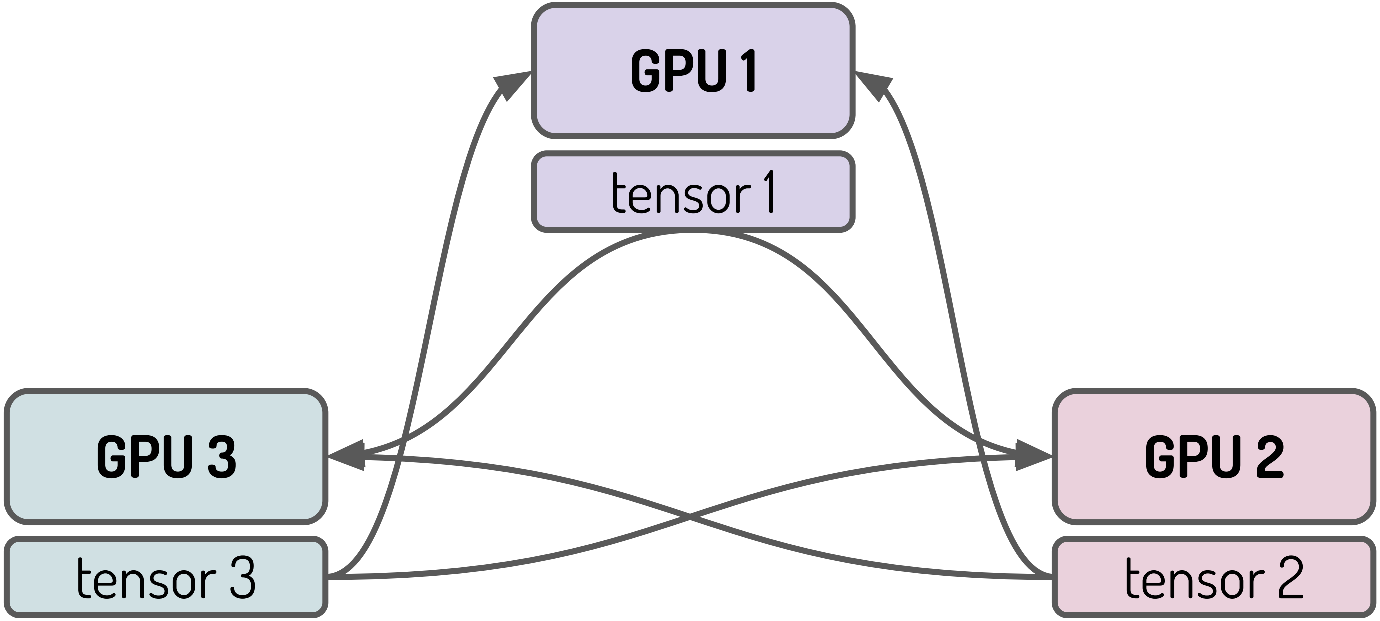
Naive All-Reduce



Naive All-Reduce

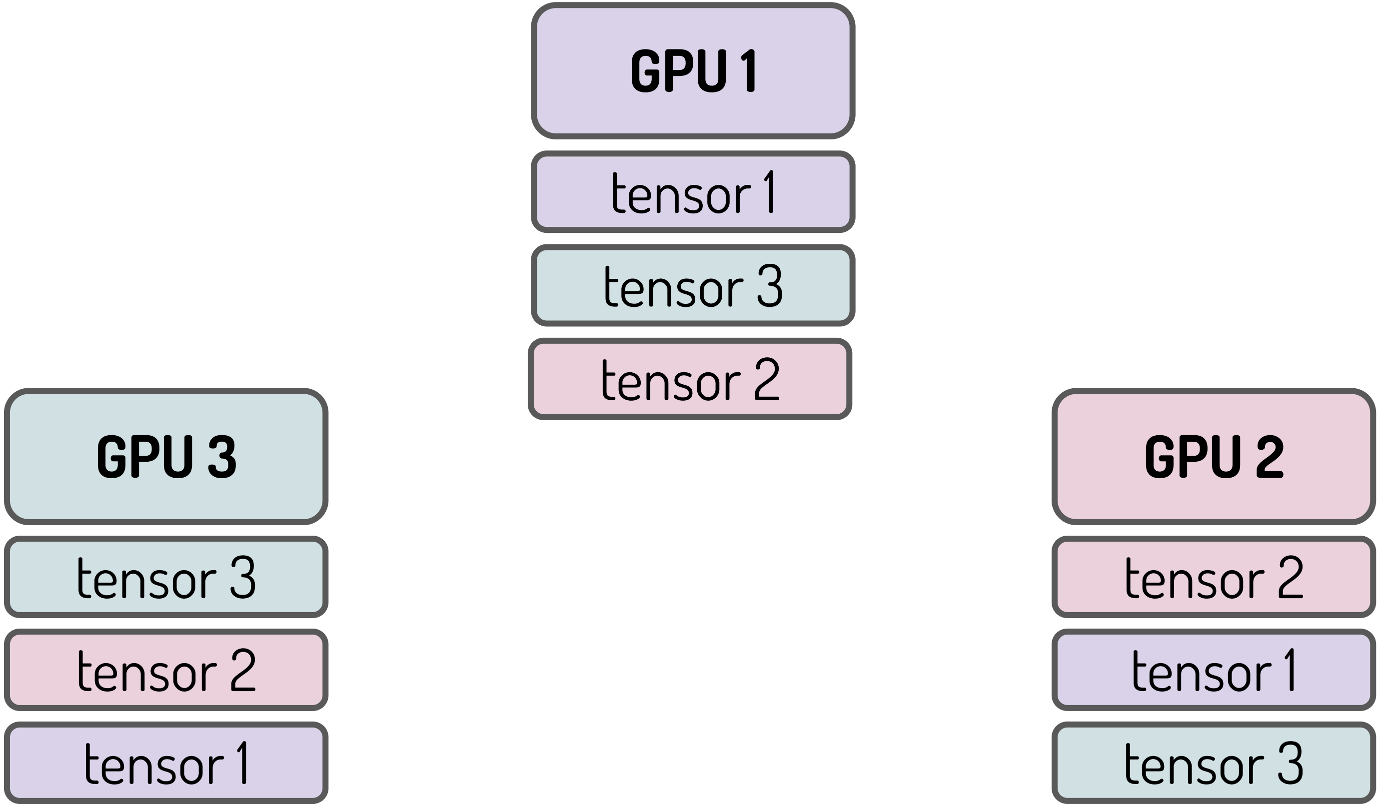


Naive All-Reduce

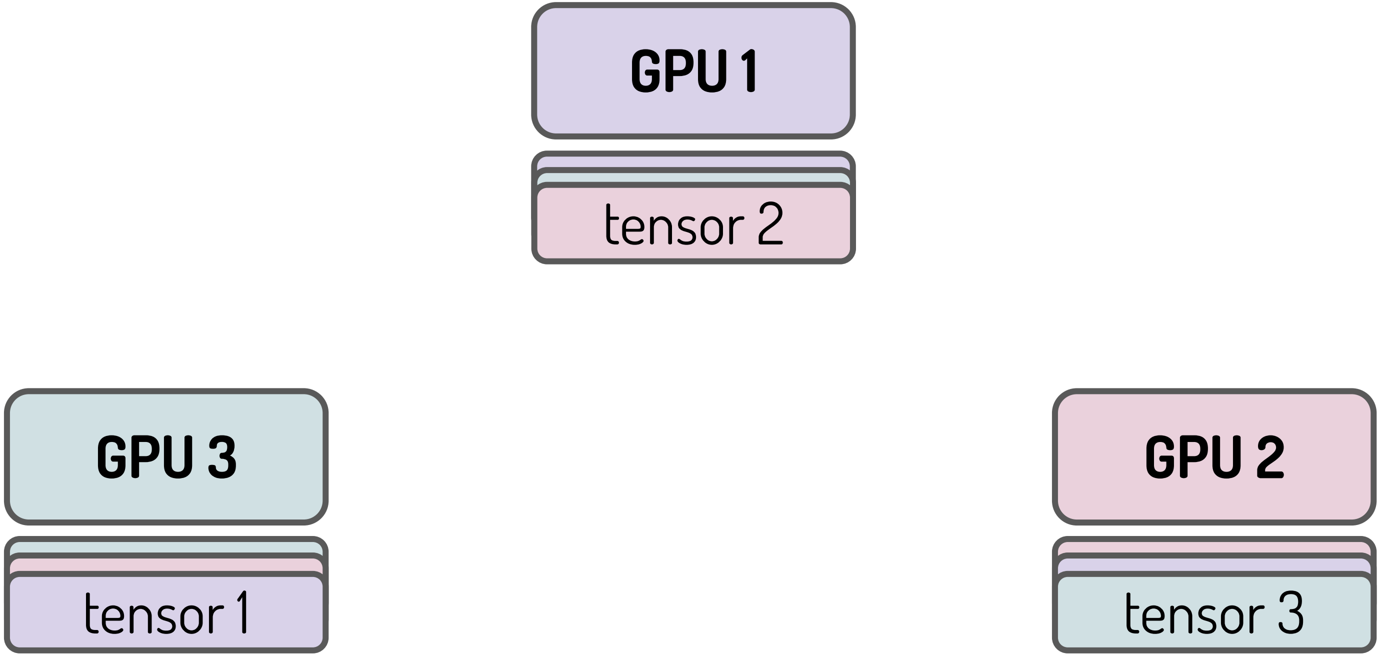


everyone sends everyone their tensor

Naive All-Reduce



Naive All-Reduce



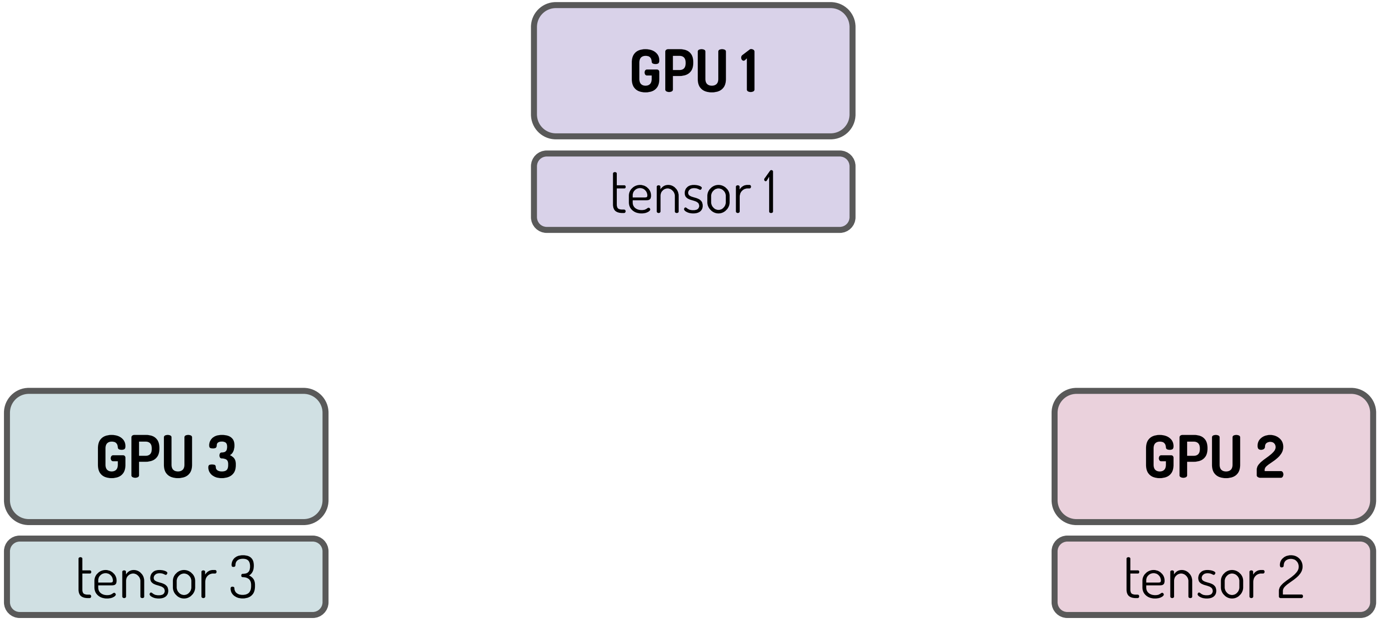
Naive All-Reduce

GPU 1

total work = p senders x (p - 1) receivers x o(n) tensor = **o(np^2)**

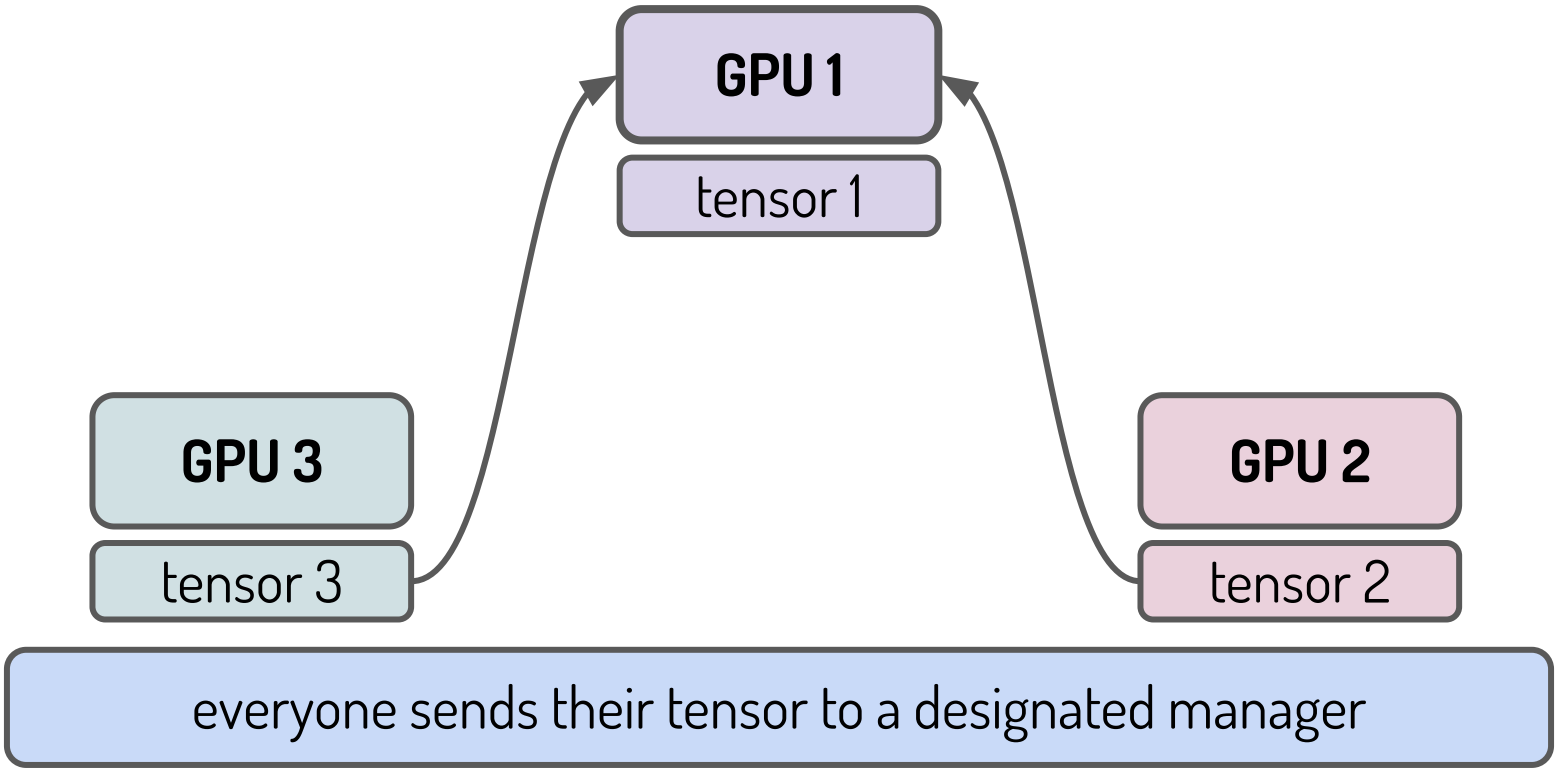
everyone does o(np) work

Another Naive All-Reduce

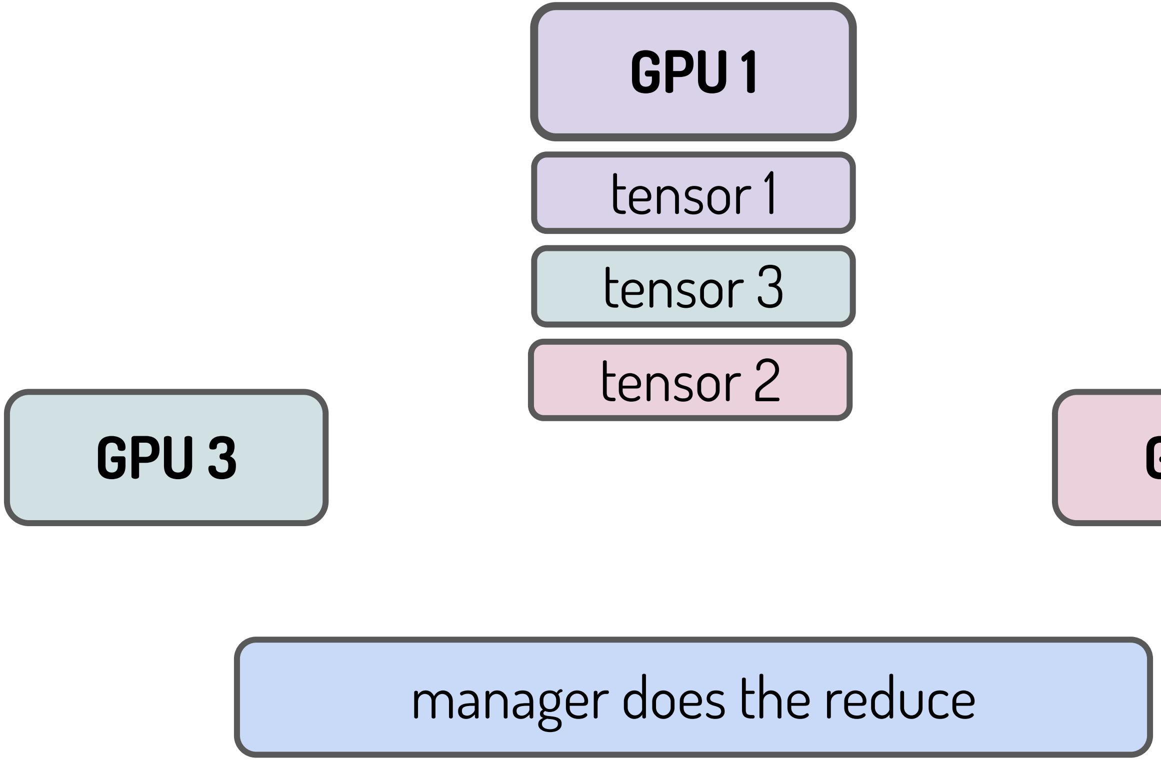


everyone sends their tensor to a designated manager

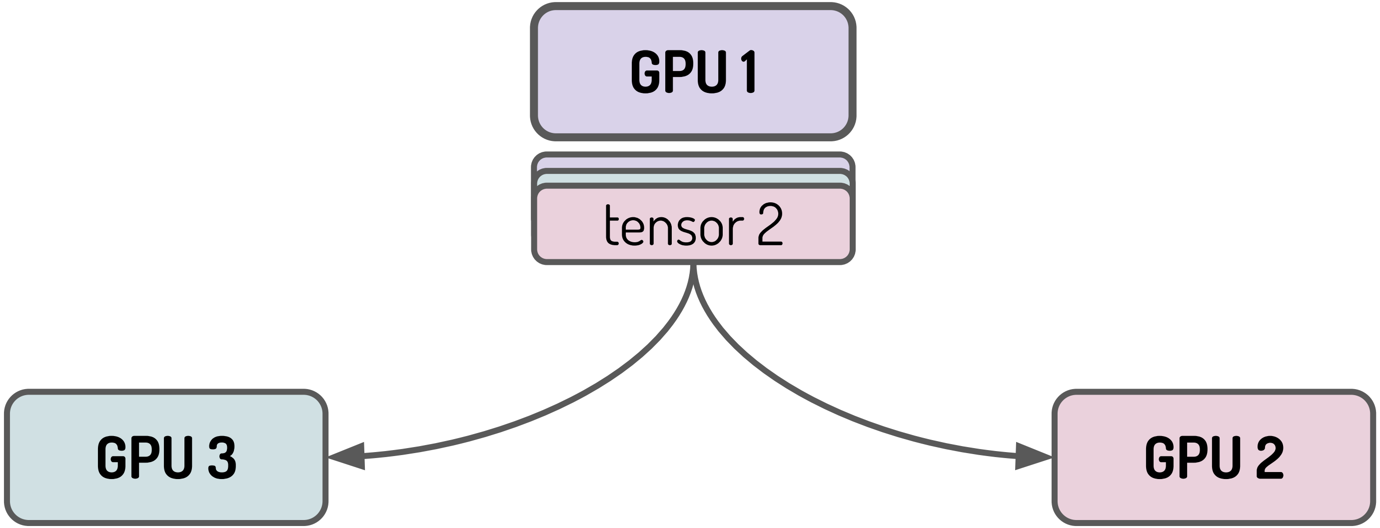
Another Naive All-Reduce



Another Naive All-Reduce

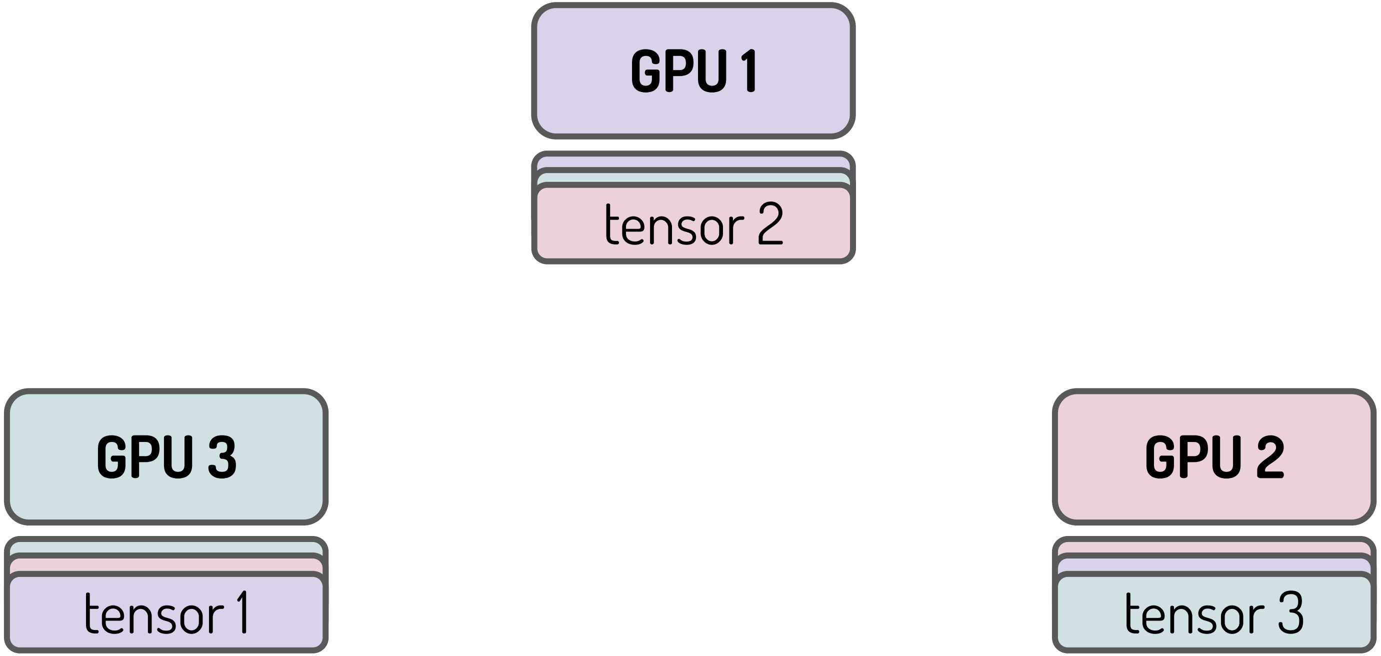


Another Naive All-Reduce

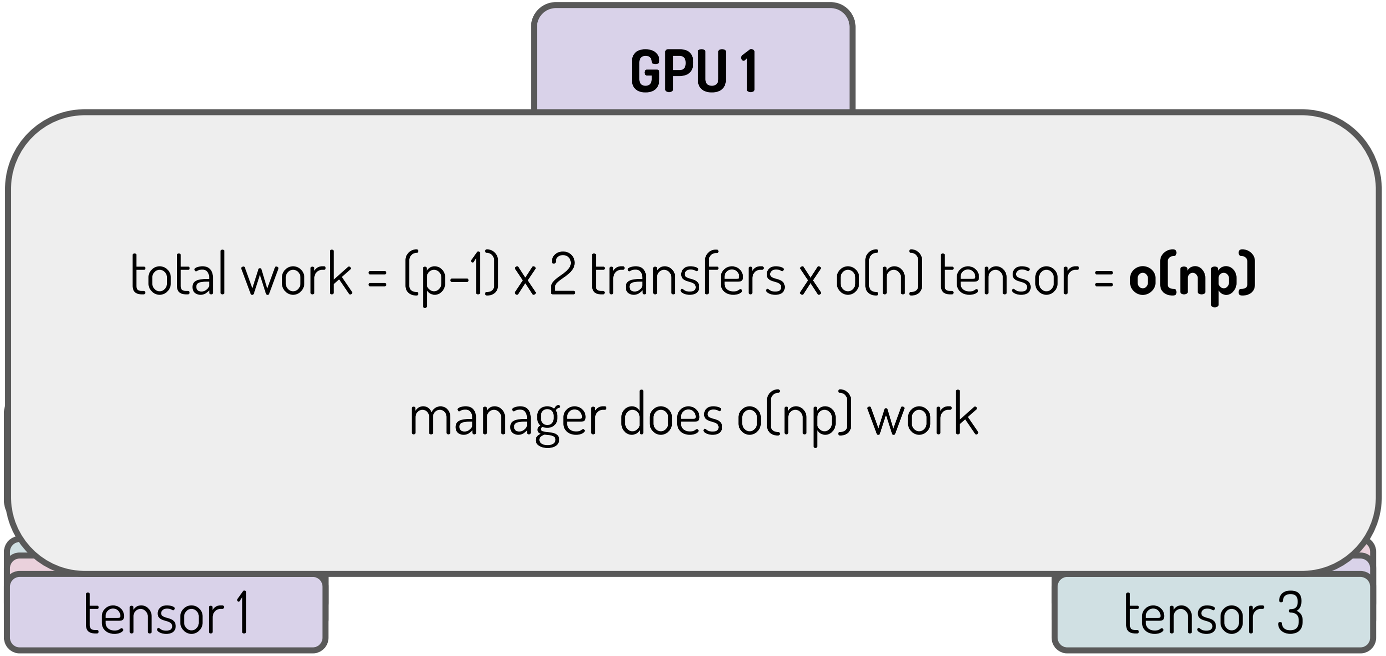


manager sends result back to everyone else

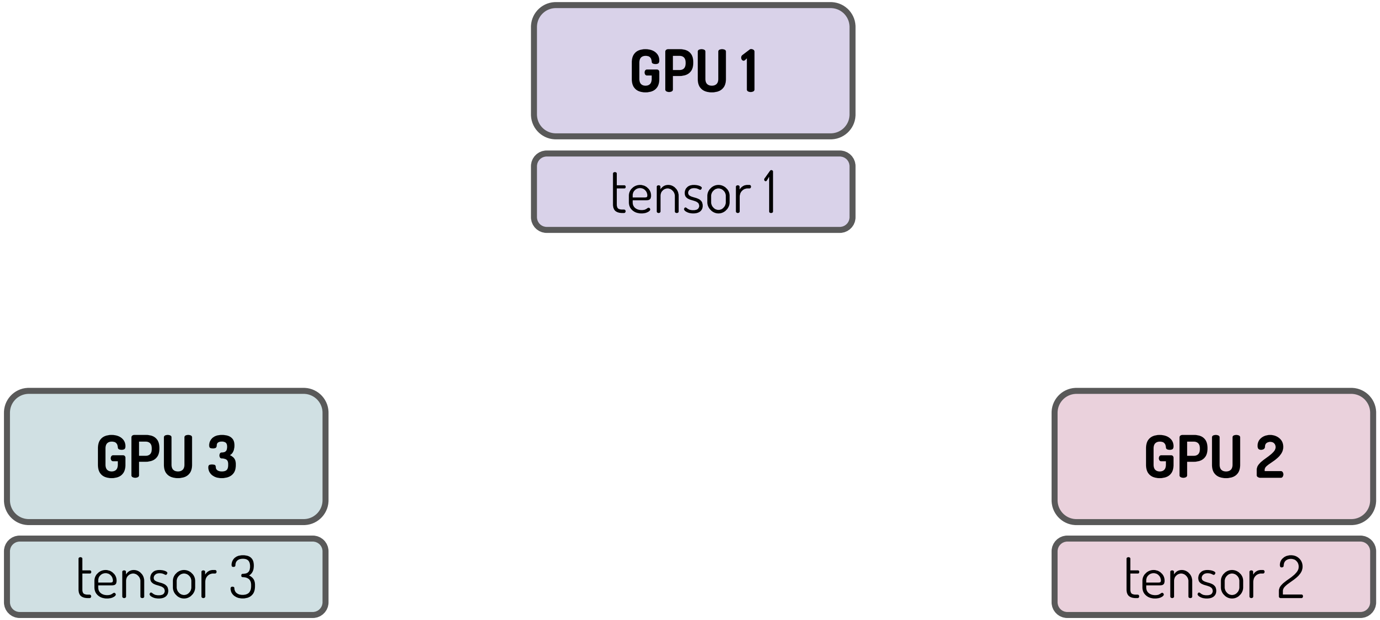
Another Naive All-Reduce



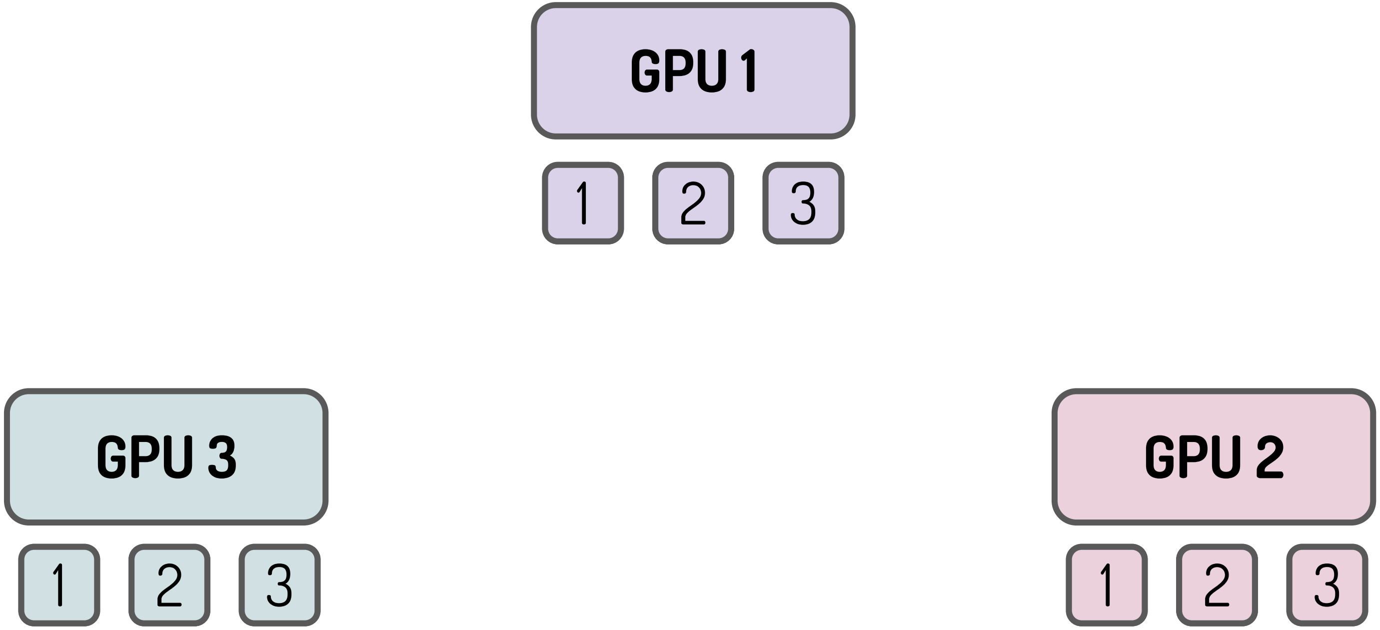
Another Naive All-Reduce



Ring All-Reduce

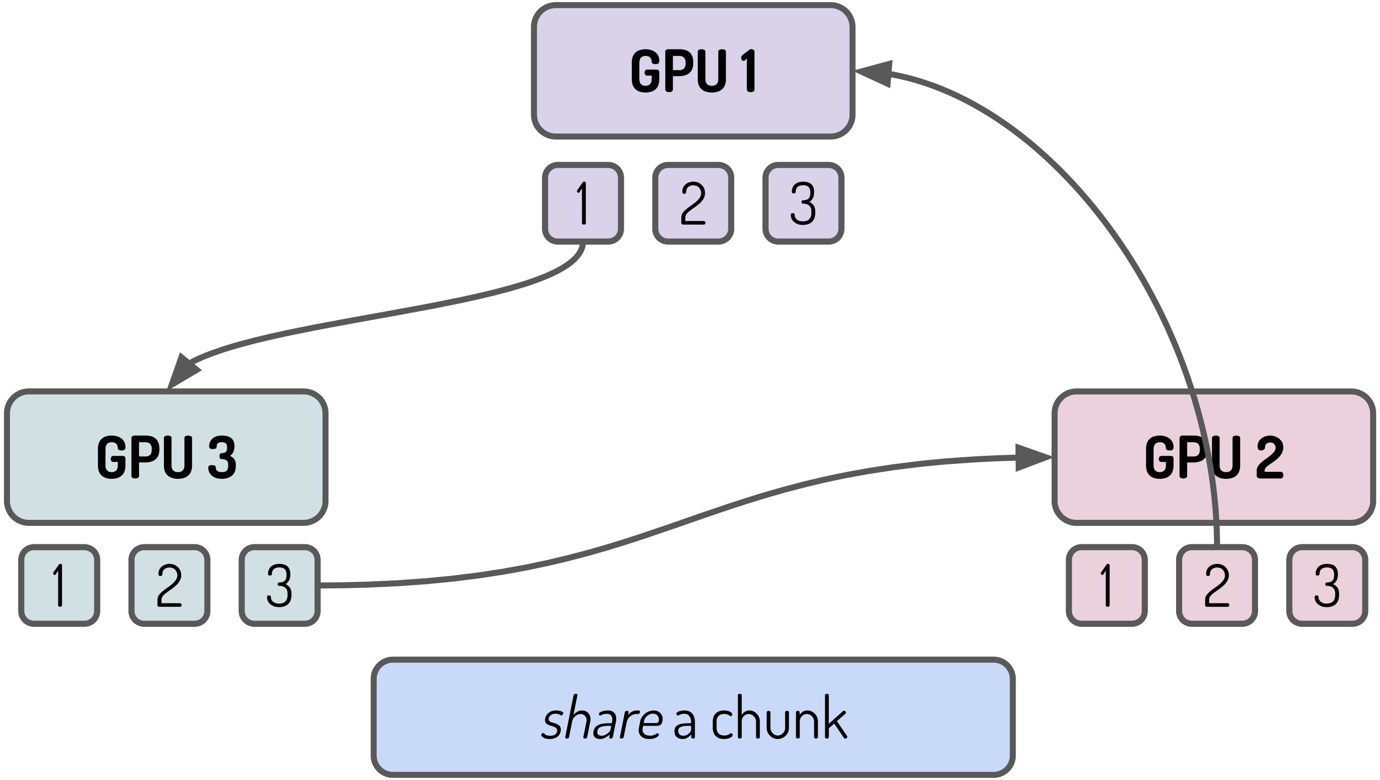


Ring All-Reduce

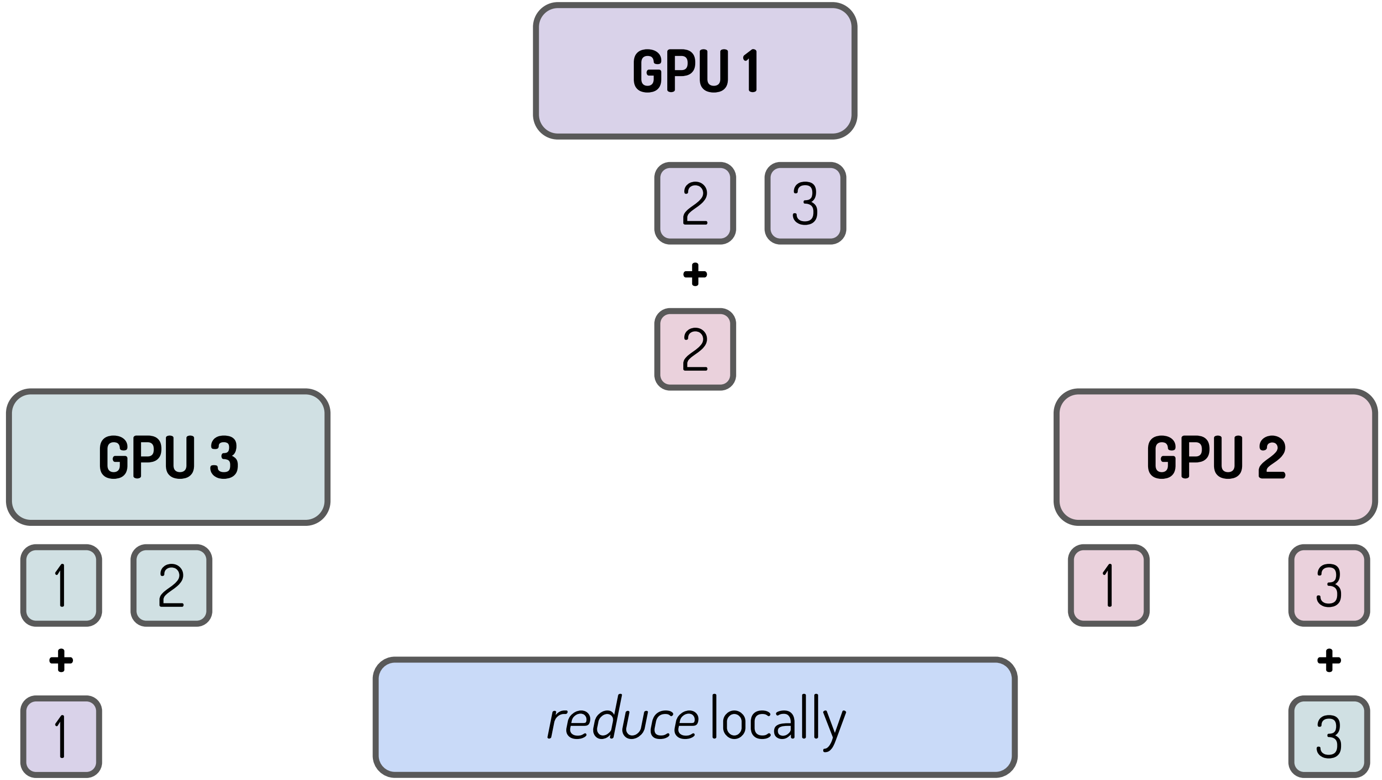


split tensor into *p* chunks

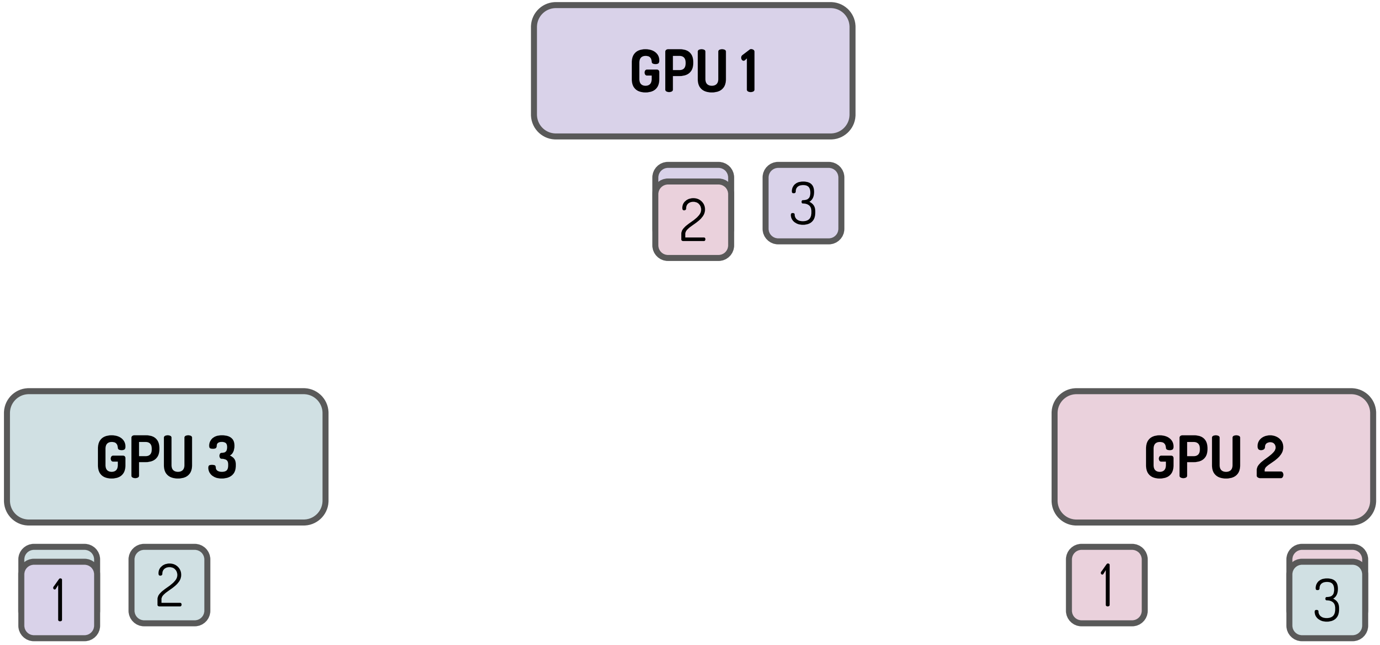
Ring All-Reduce



Ring All-Reduce

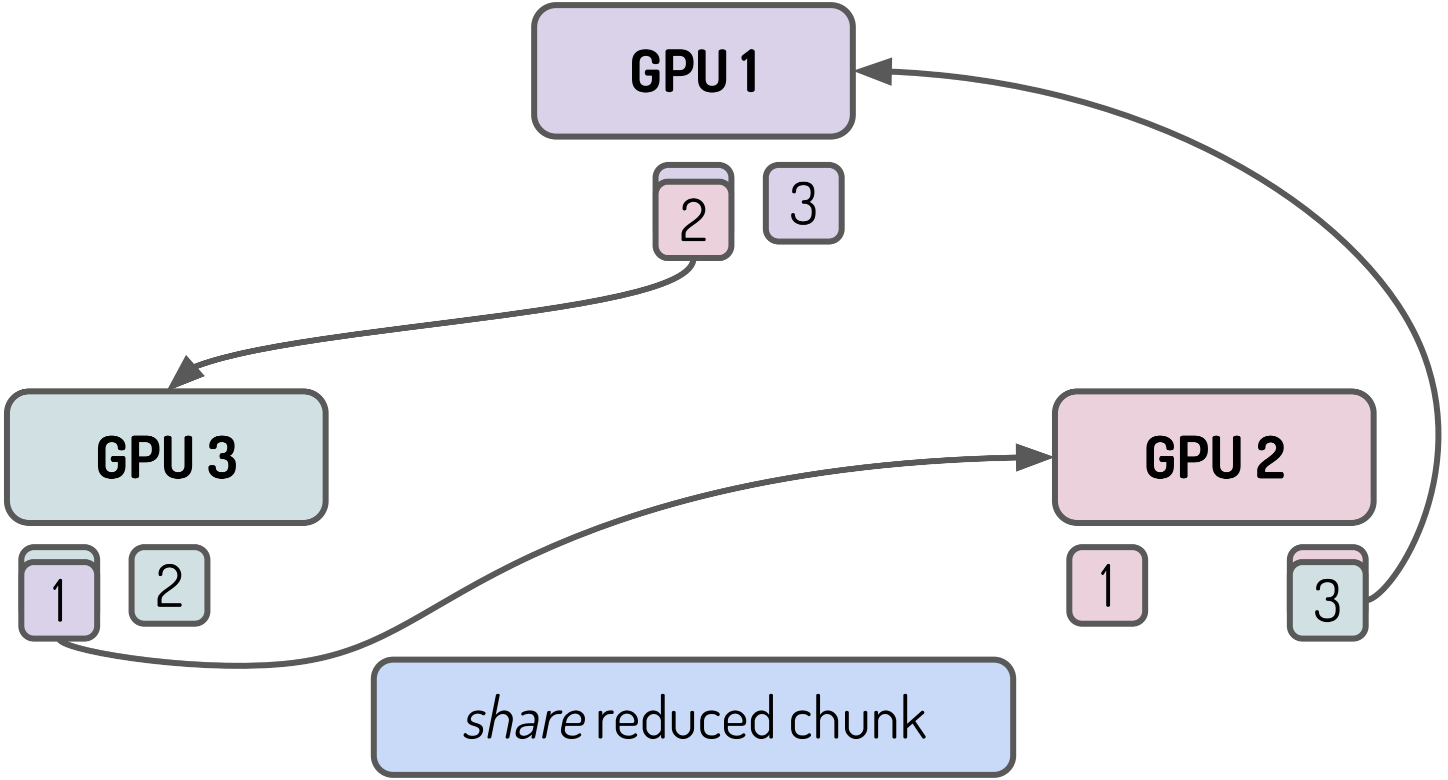


Ring All-Reduce

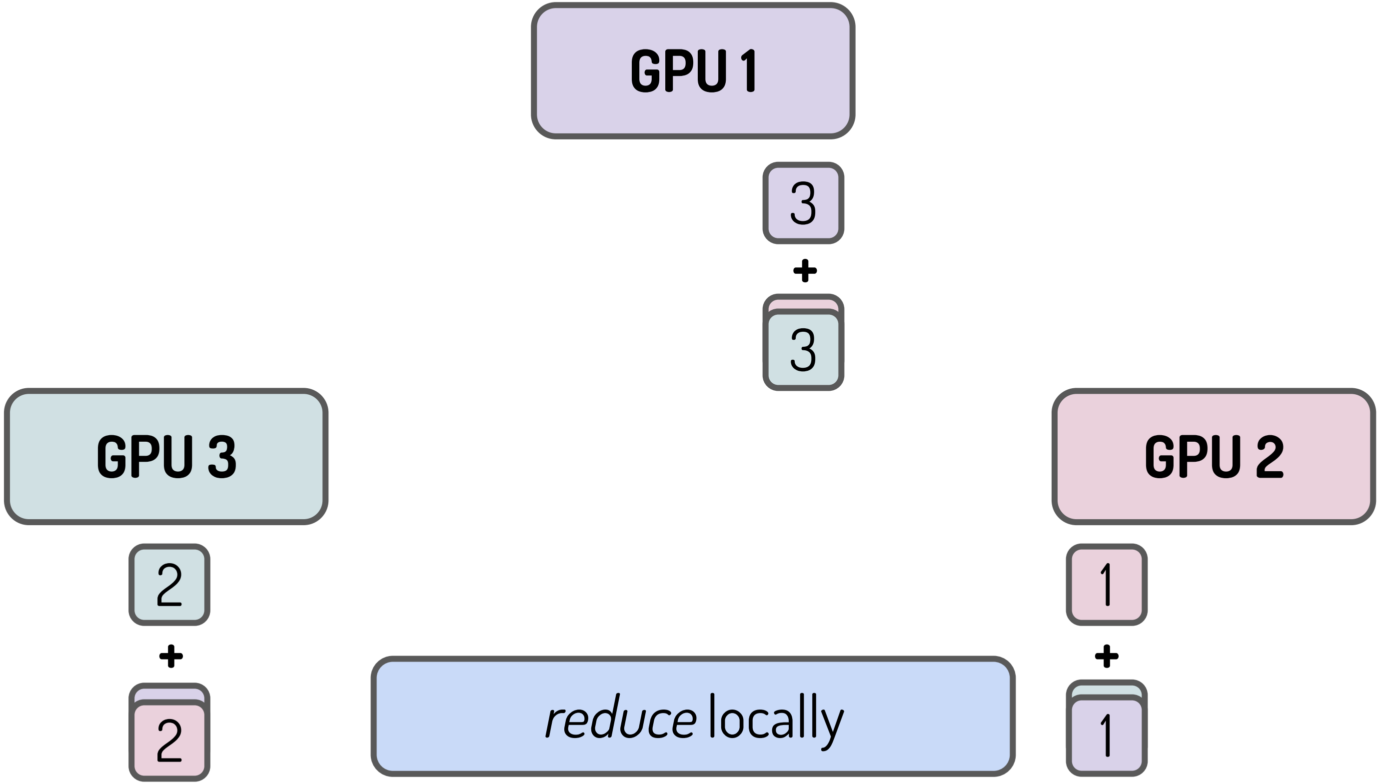


reduce locally

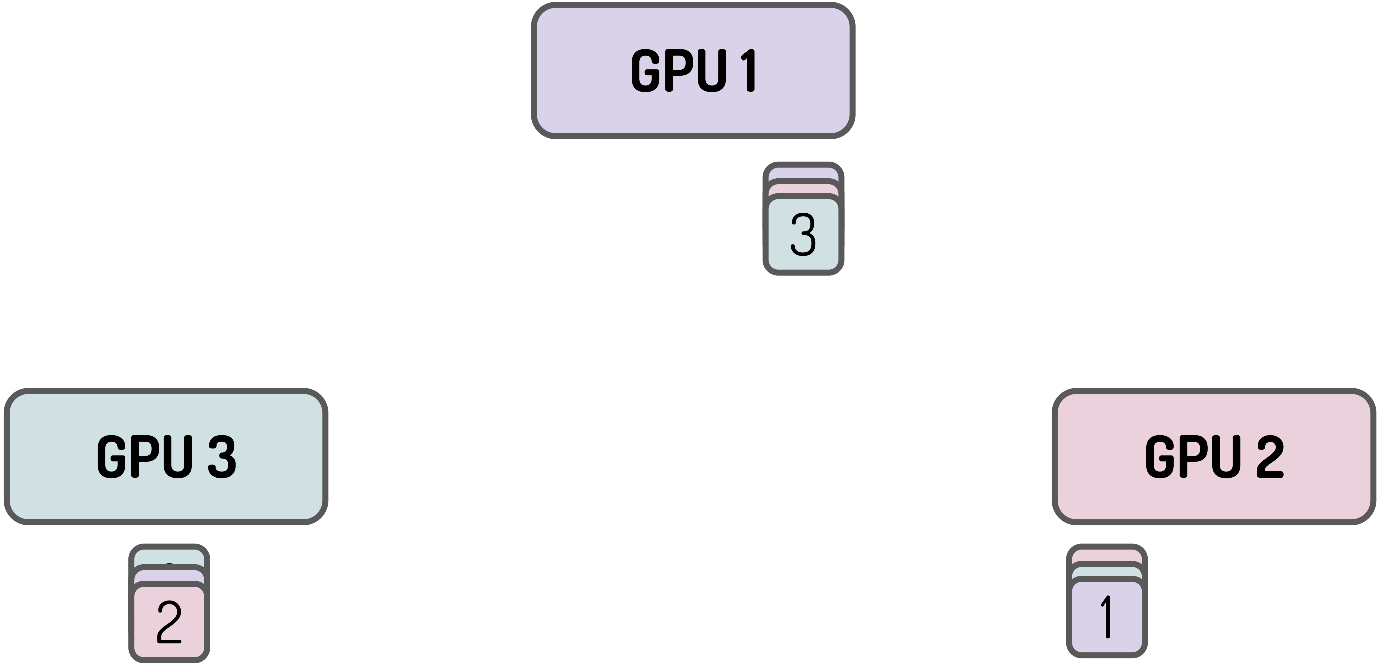
Ring All-Reduce



Ring All-Reduce

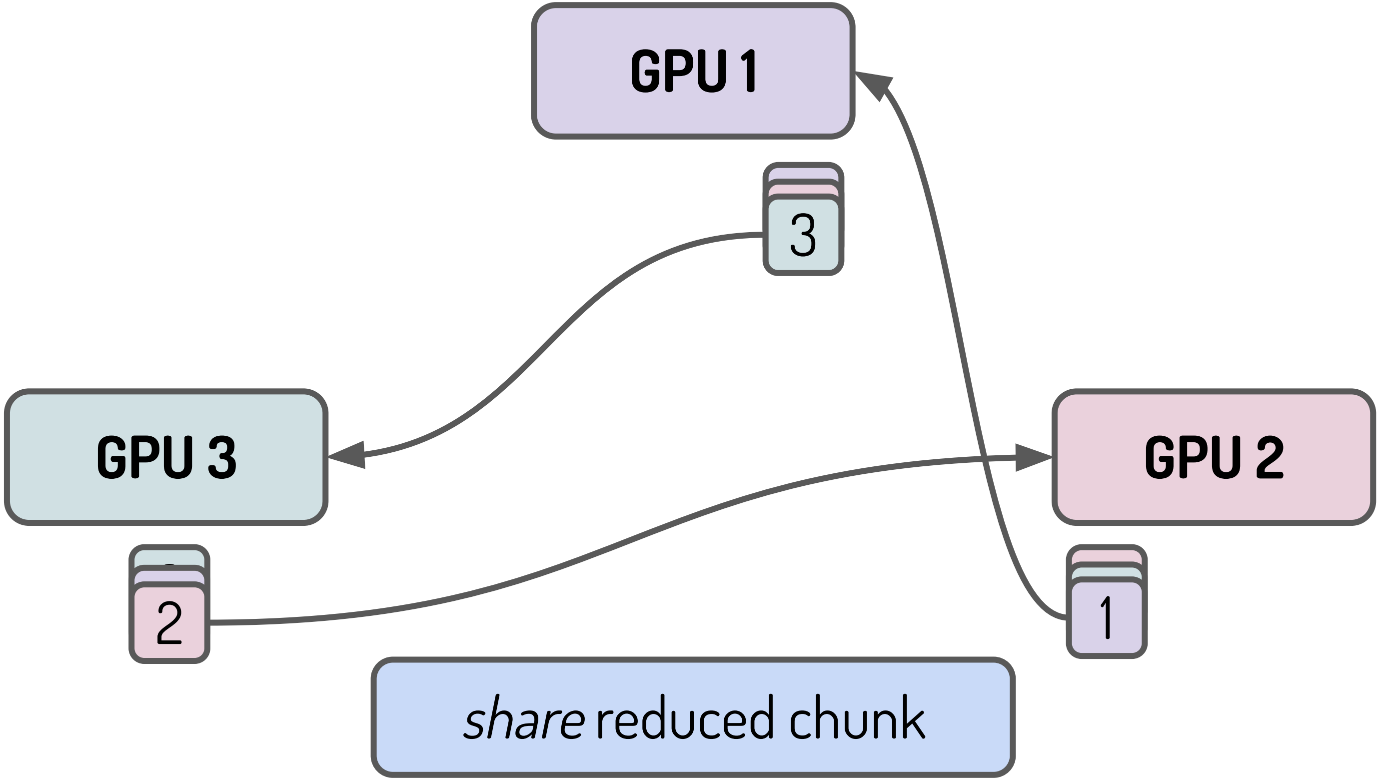


Ring All-Reduce

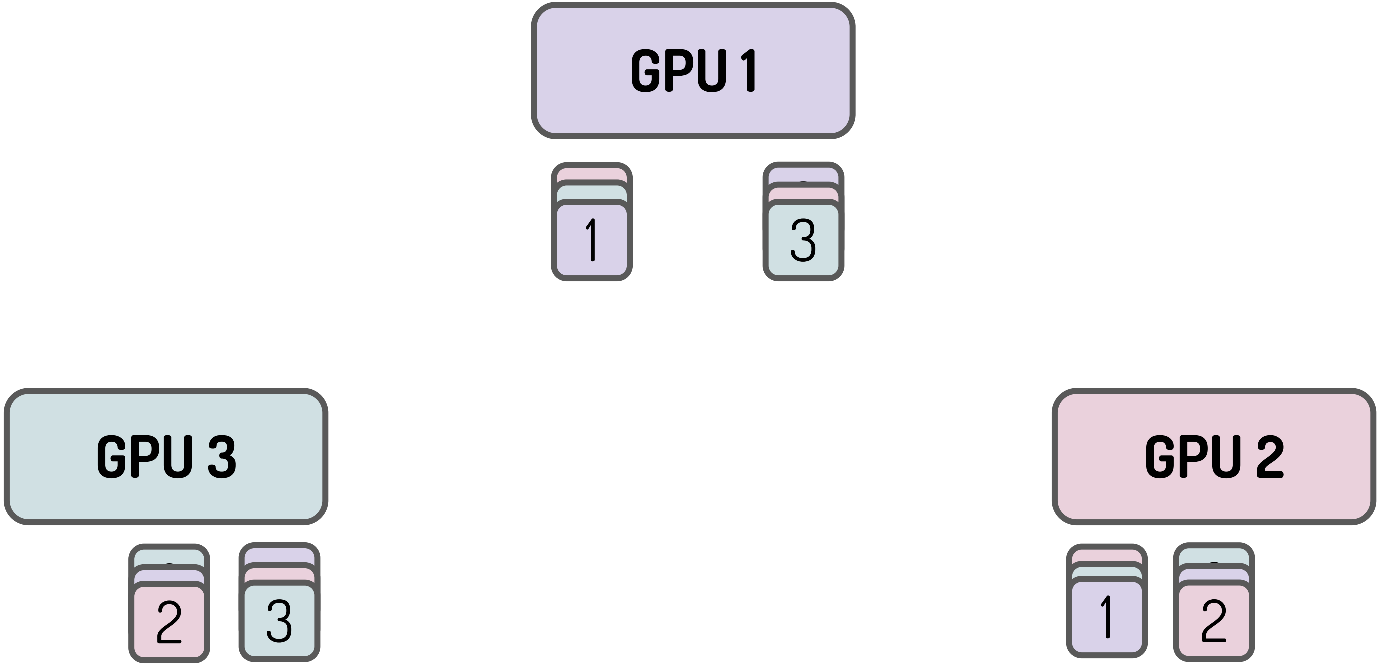


everyone has a chunk of the result after *(p-1)* share-reduce steps

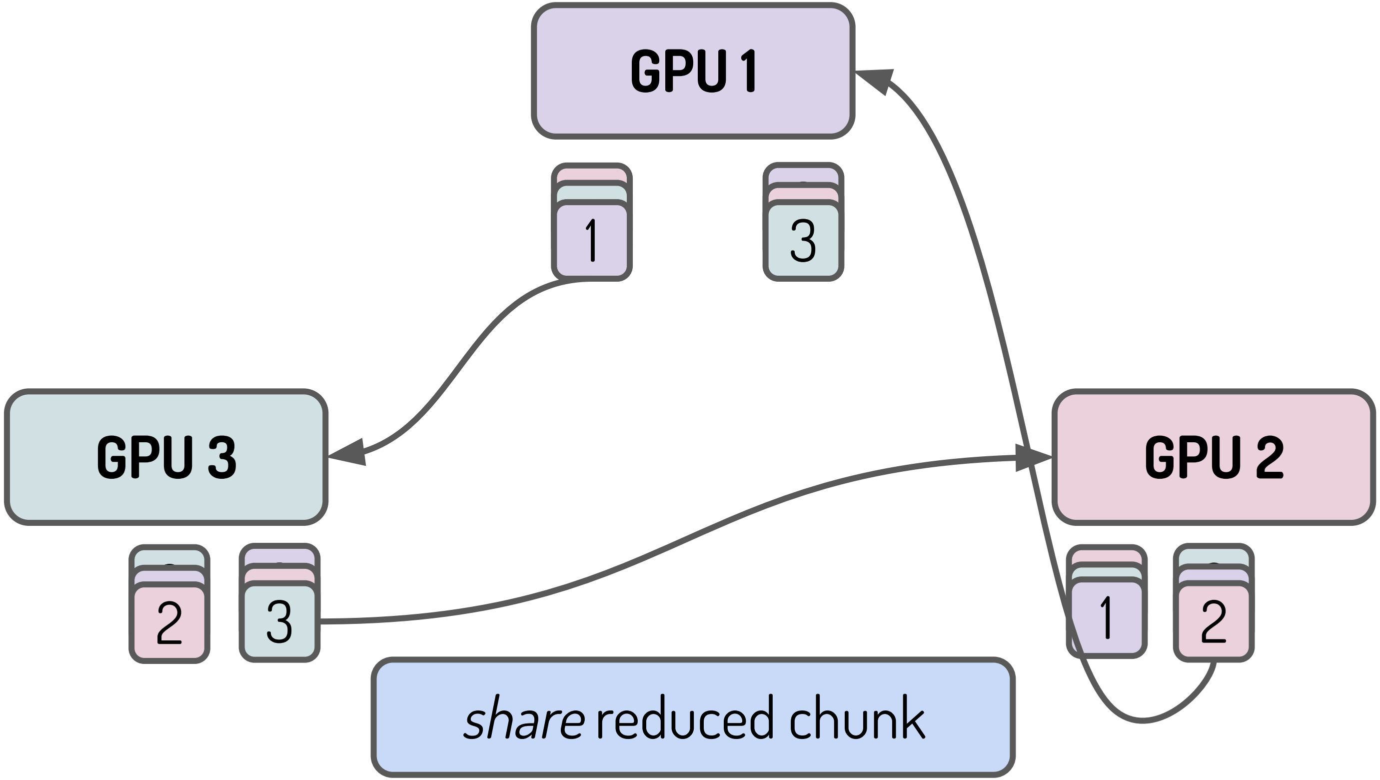
Ring All-Reduce



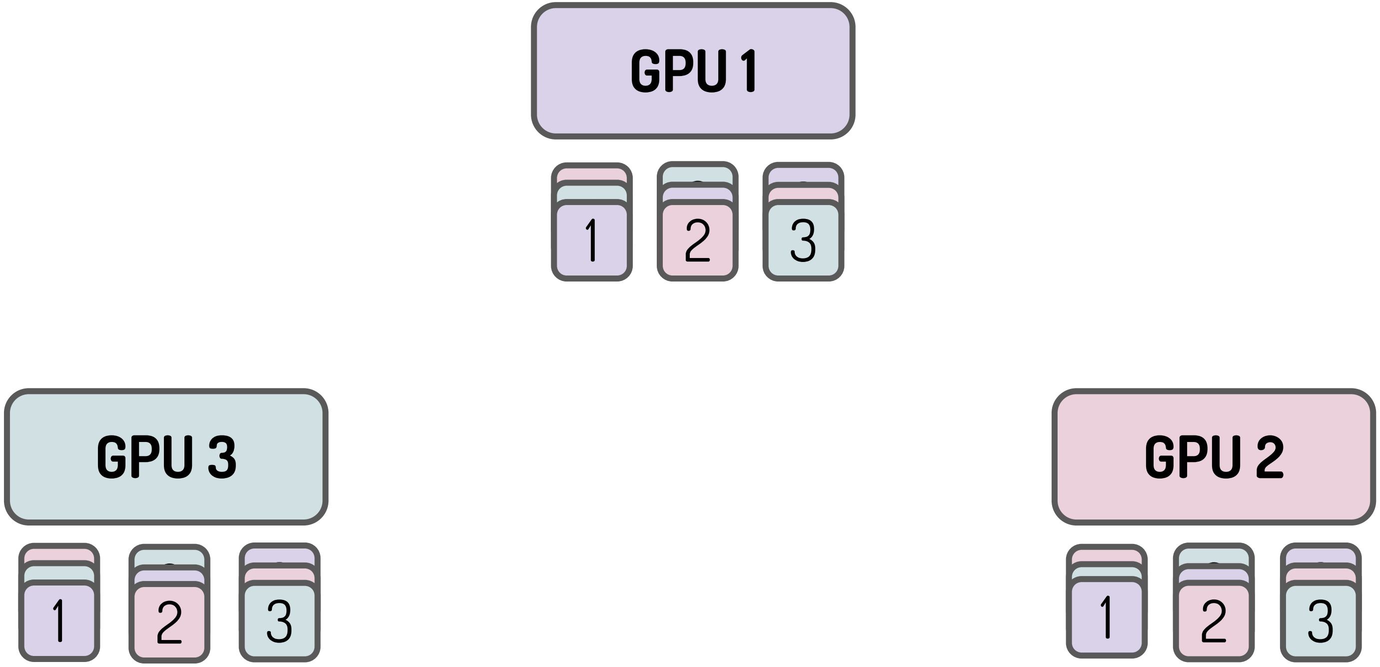
Ring All-Reduce



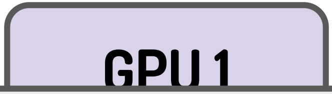
Ring All-Reduce



Ring All-Reduce

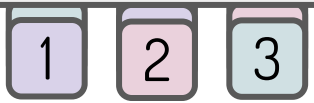


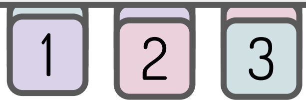
Ring All-Reduce



total work = p senders x 1 receiver x o(n/p) tensor x (p-1) rounds x 2 phases = **o(np)**

everyone does equal o(n) work (independent of p!)

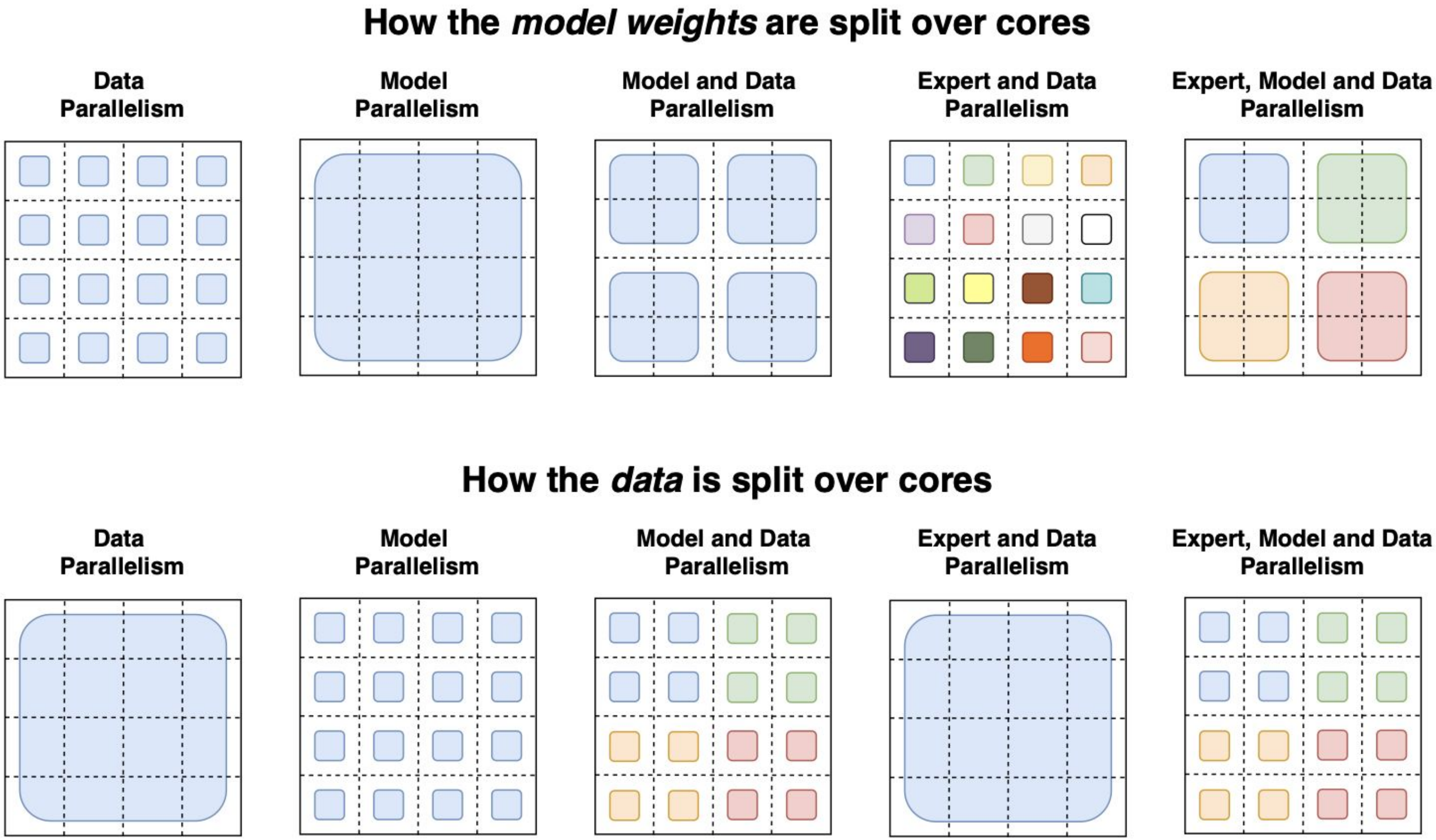




Ring All-Reduce

nVidia Collective Communications Library (nccl)

Horovod distributed training



Credit: Fedus et al. (Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity)

Large-Scale Data in Language Modeling

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Text

Credit: Brown et al. (Language Models are Few-Shot Learners)

Large-Scale Data in Visual Learning

3. The JFT-300M Dataset

We now introduce the JFT-300M dataset used throughout this paper. JFT-300M is a follow up version of the dataset introduced by [7, 17]. The JFT-300M dataset is closely related and derived from the data which powers the Image Search. In this version, the dataset has 300M images and 375M labels, on average each image has 1.26 labels. These images are labeled with 18291 categories: *e.g.*, **1165** type of animals and **5720** types of vehicles are labeled in the dataset. These categories form a rich hierarchy with the maximum depth of hierarchy being **12** and maximum number of child for parent node being **2876**.

Credit: Sun et al. (Revisiting Unreasonable Effectiveness of Data in Deep Learning Era)

Feed Data Fast

Columnar Data	Apache Parquet (Disk)	Apache Arrow (In-Memory)
Parallel Workers	Apache Spark	Multiprocessing	Multithreading
Libraries	HuggingFace Datasets	Uber Petastorm	Tensorflow Datasets

Warning: primary memory is a bottleneck!

Lots More To Read

RAY

Exploring the Limits of
Weakly Supervised Pretraining

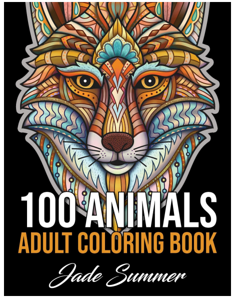
MegatronLM: Training Billion+ Parameter Language Models Using GPU Model Parallelism

1

2

9

4



5

3

6

7

8

10

Machine Learning Systems Design

Next class: Model evaluation