Programming for Data Science at Scale

Distributed Machine Learning

THE UNIVERSITY *of* EDINBURGH

Amir Noohi, Fall 2024

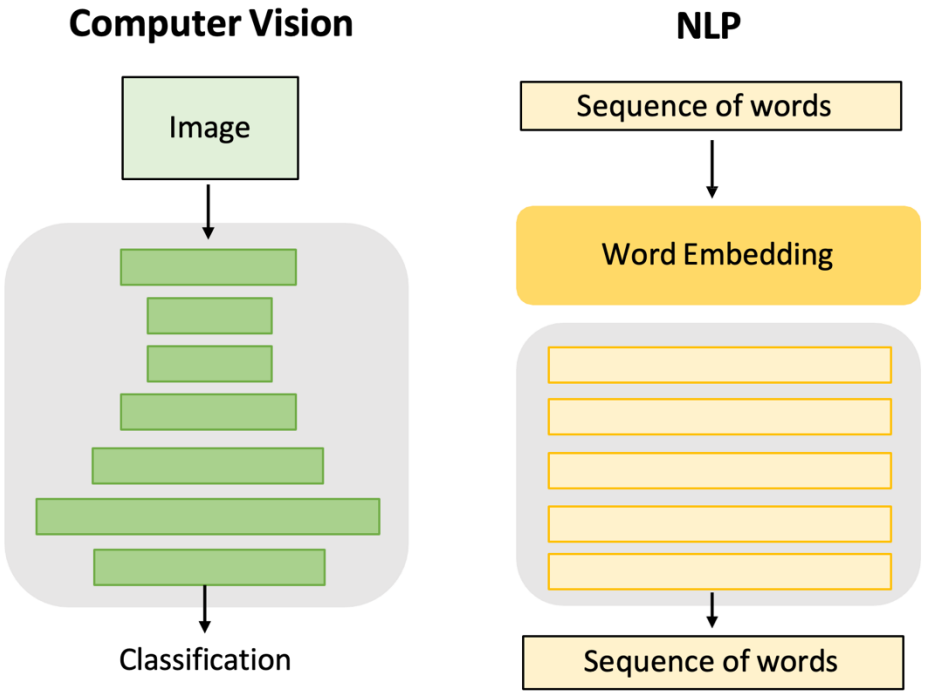
Many ML applications are emerging

Deep neural networks

- Better accuracy
- High computation cost
- **Gradient-based** training

Diverse applications

- Natural language processing
- Deep reinforcement learning
- Graph neural networks
- …



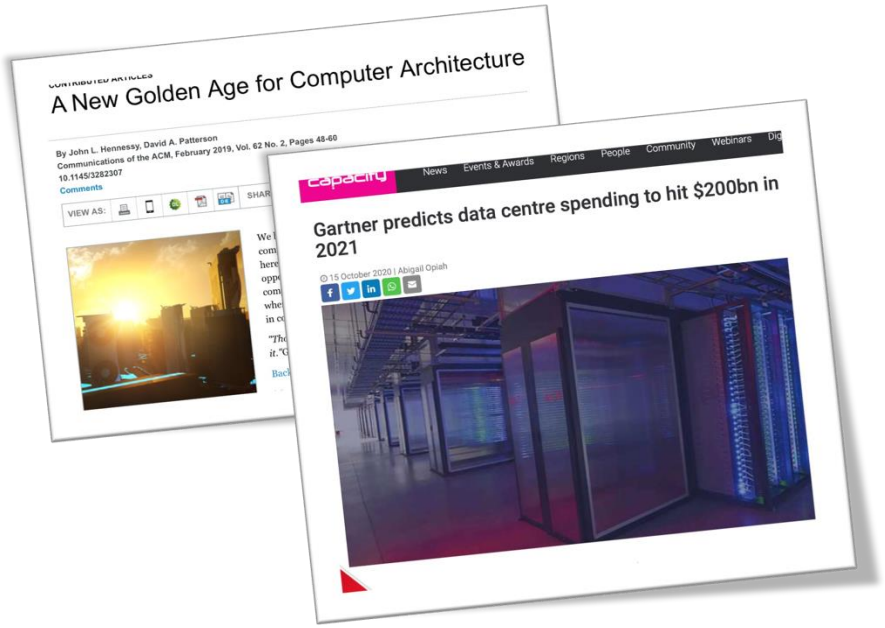
Massive computational power is available

Heterogeneous processors

- CPUs, GPUs and TPUs
- 10 – 100x acceleration

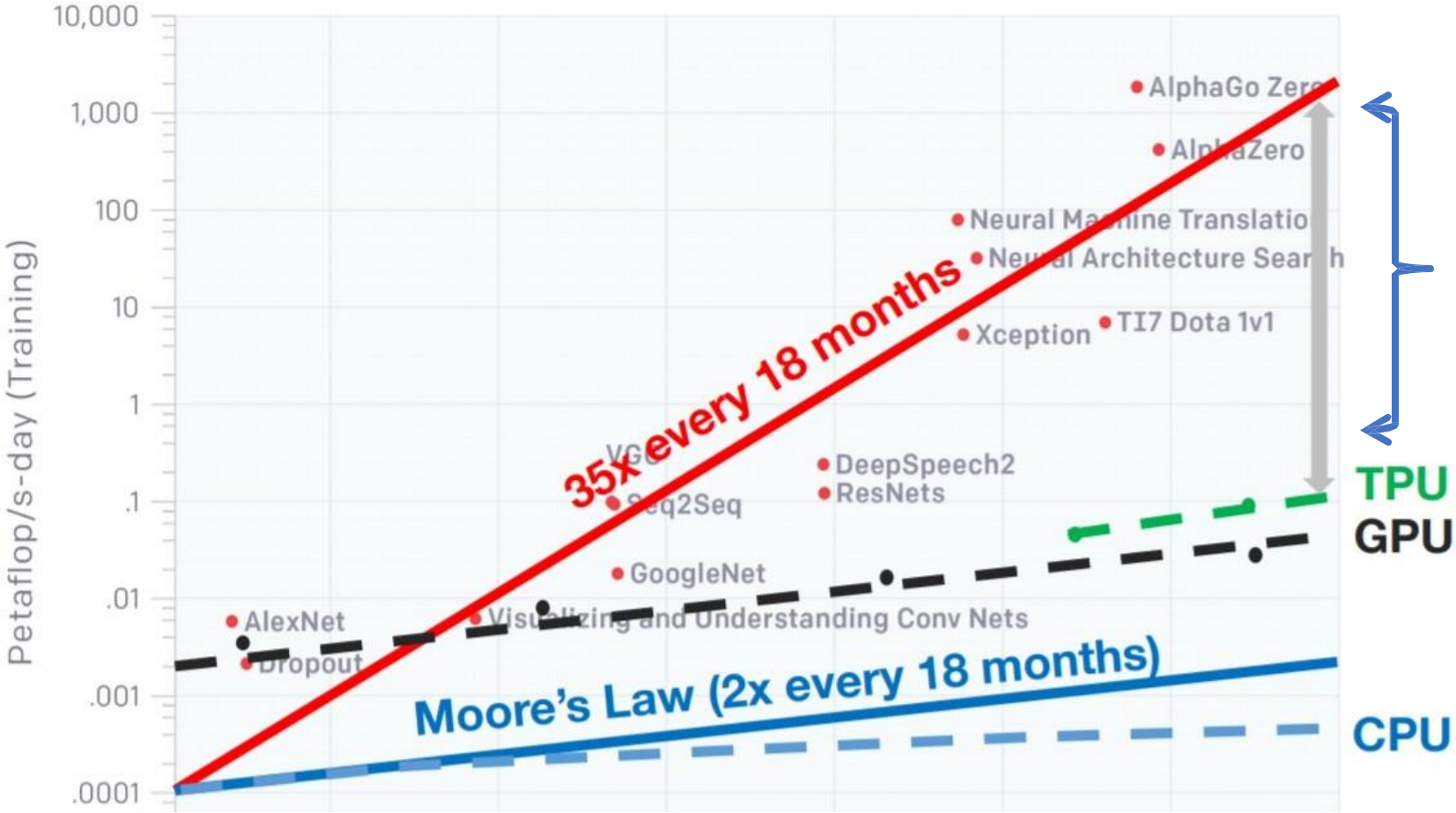
Global data centres

- Easy access to PB-scale data
- 100,000s machines



Three key factors that drive AI booming: Algorithms, Hardware, Data

Motivation behind Distributed ML



ML frameworks: A new category of system software

	Neural Networks	Automatic Differentiation	Un/semi-structured data management	Training & Inference	Heterogenous Processors	Distributed Execution
Neural network libraries (Theano, Caffe)	✓	✓	✗	✗	✓	✗
Data parallel systems (Spark, Giraph)	✗	✗	✗	✗	✗	✓
ML framework (PyTorch, TensorFlow)	✓	✓	✓	✓	✓	✓

System architectures of ML frameworks

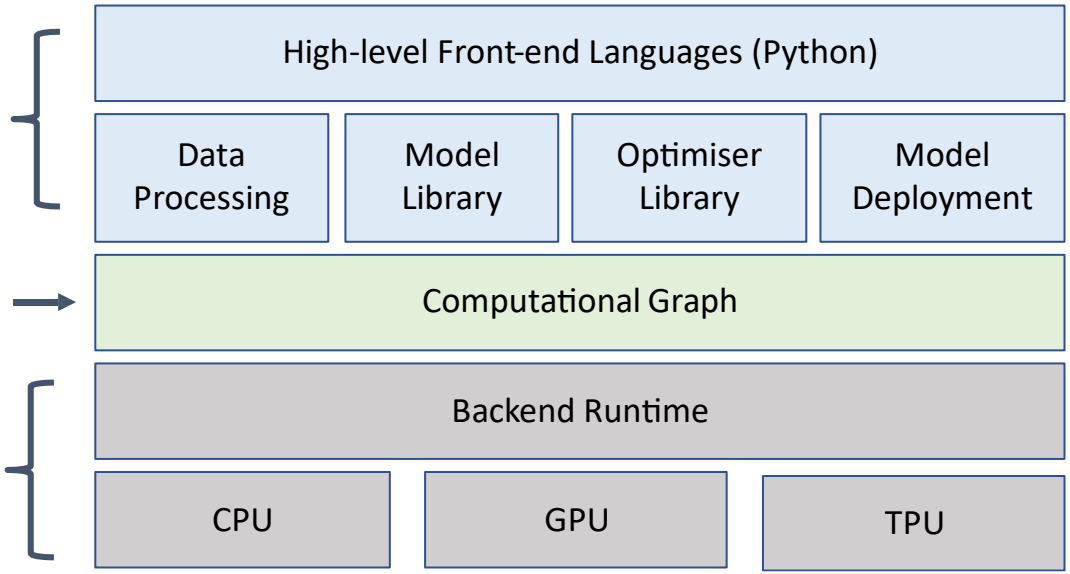
Design Goals

1. Programming abstraction: Supporting ML in different applications

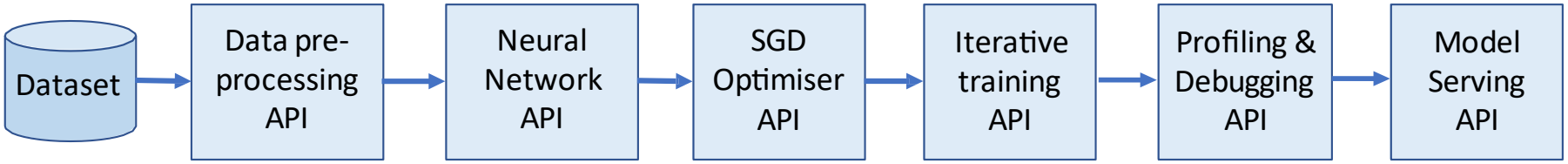
2. Execution engine: Enable gradient-based computation & parallelise computation

3. Hardware runtime: utilise all heterogeneous processors

Architecture



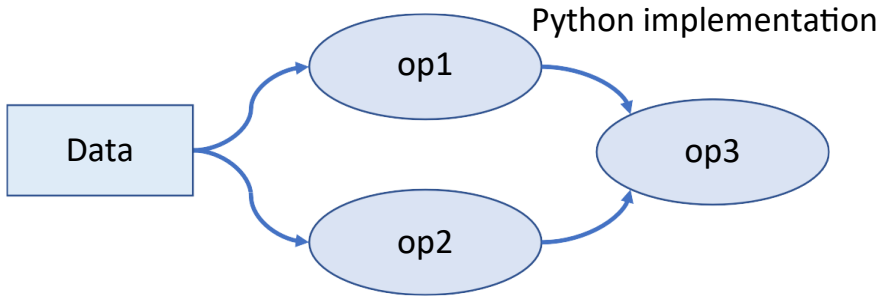
ML framework programming abstraction



An unified programming abstraction for different ML applications (DNNs, GNNs, DRLs, …)

Front-end language: Python

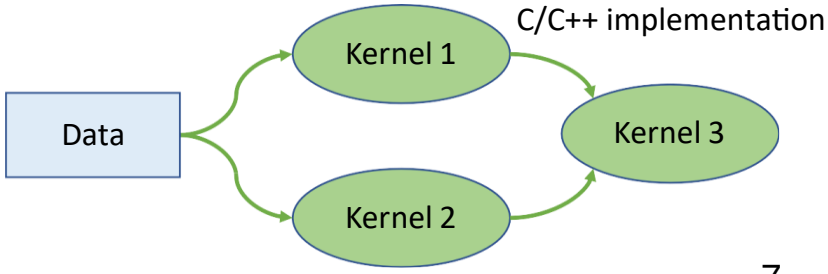
- Simple and flexible
- Poor performance
- Global Interpreter Lock (GIL)



Equivalent Back-end Graph

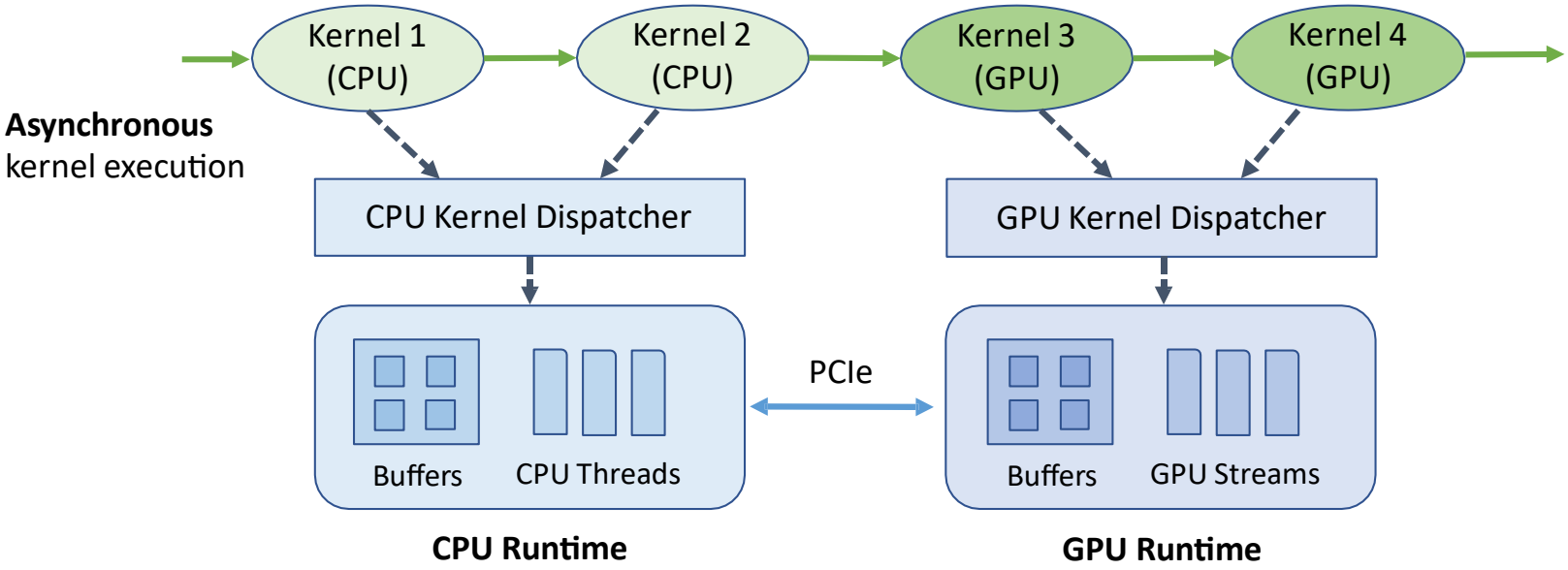
Back-end language: C/C++

- Hardware-friendly
- Excellent performance

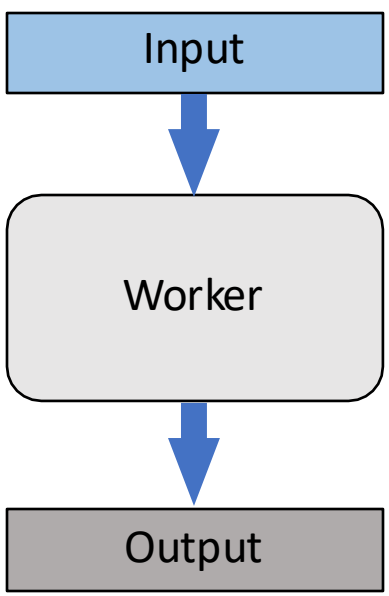


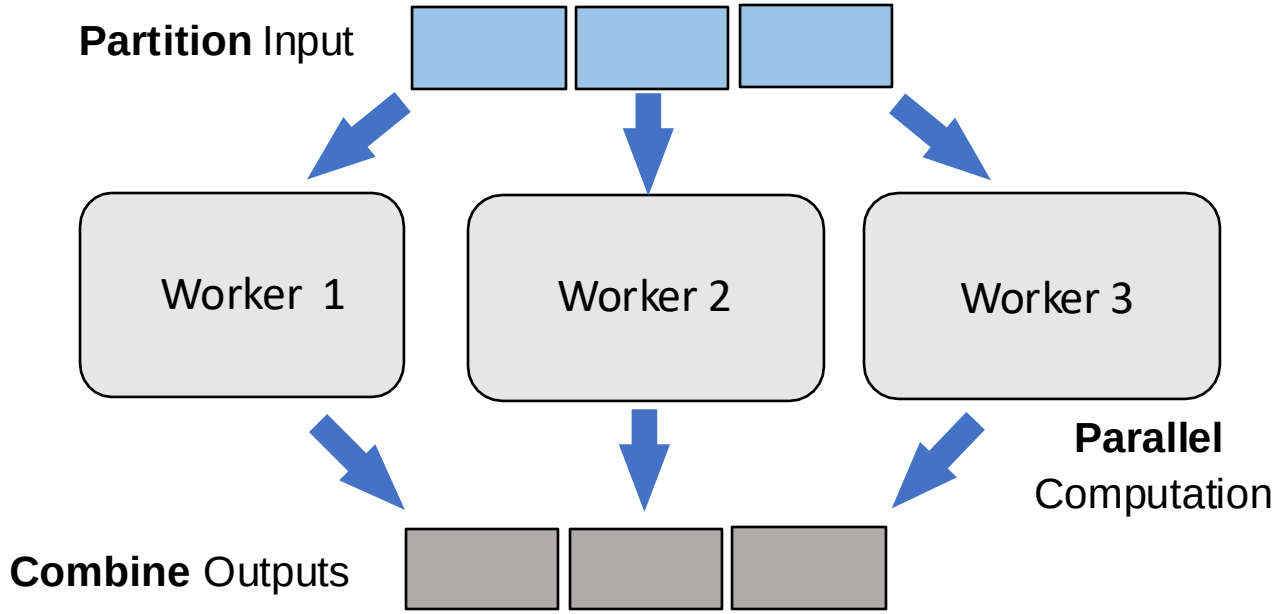
Using heterogenous processors

Operators in ML models have execution kernels for CPUs and GPUs



Distributed Multi worker Execution





(a) Single-worker Execution

(b) Distributed Multi-worker Execution

Why Distributed ML Systems?

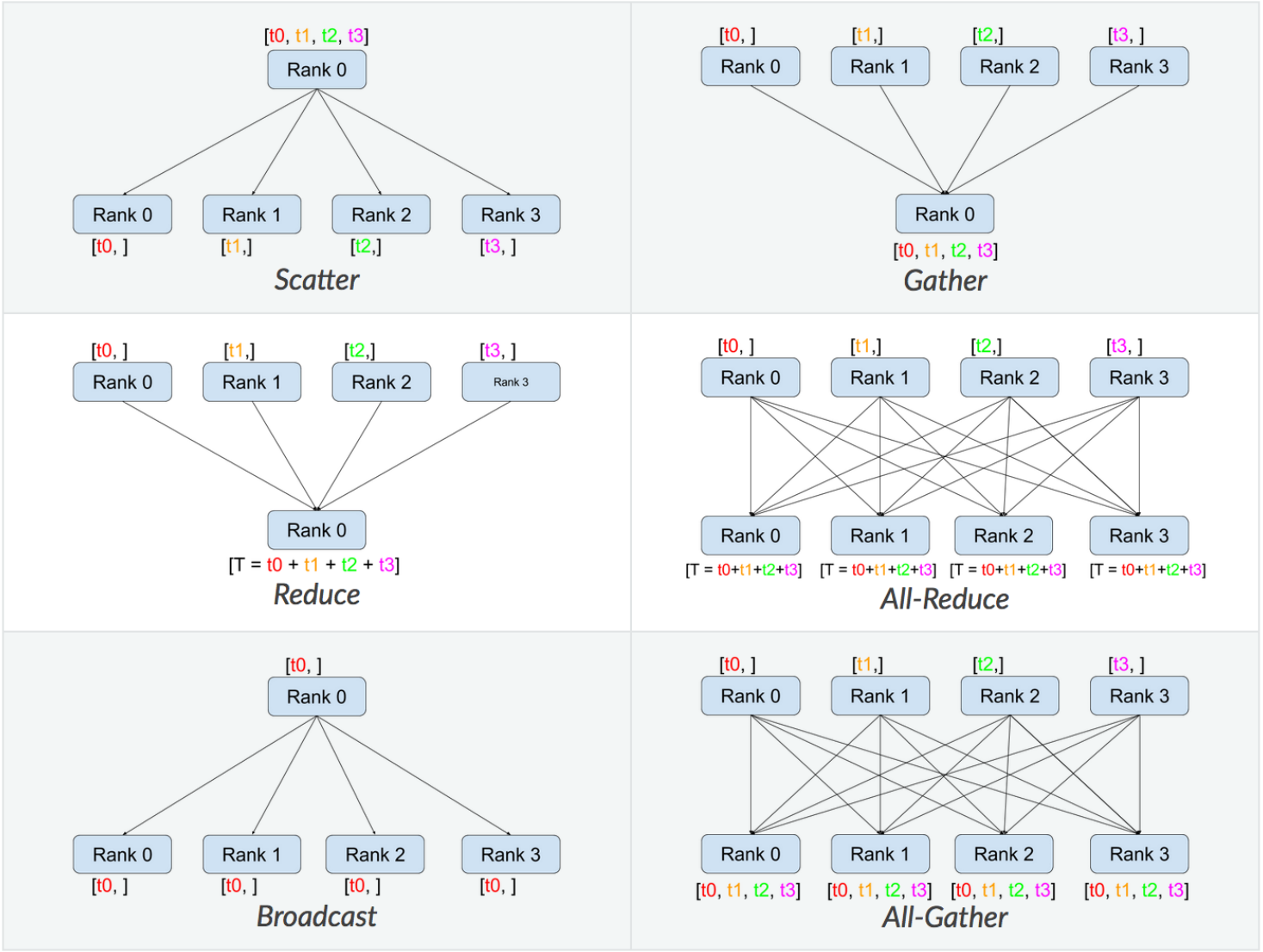
- Performance
  - Reducing the time to complete a data epoch
- Memory wall
- Economy
  - Multiple commodity servers, instead of a single expensive high-end server
- Hardware failure tolerance

How to make ML distributed?

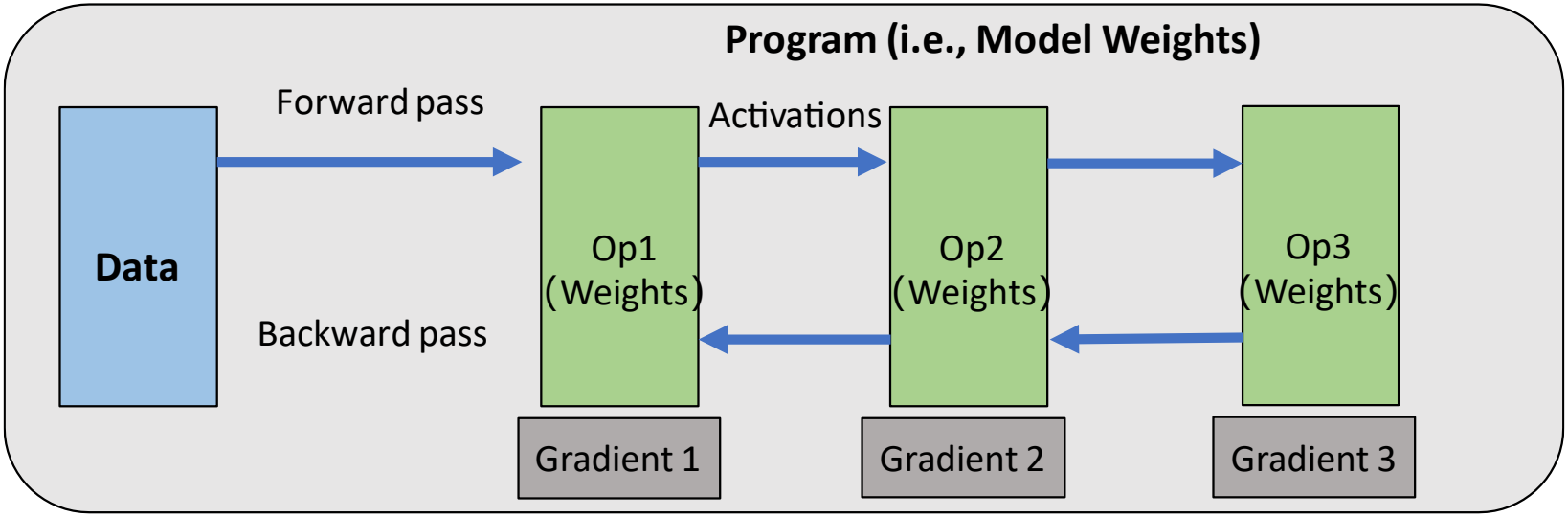
- Communication Backends:
  - NCCL
  - GLOO
  - MPI

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✘	✓	?	✘	✓
recv	✓	✘	✓	?	✘	✓
broadcast	✓	✓	✓	?	✘	✓
all_reduce	✓	✓	✓	?	✘	✓
reduce	✓	✘	✓	?	✘	✓
all_gather	✓	✘	✓	?	✘	✓
gather	✓	✘	✓	?	✘	✓
scatter	✓	✘	✓	?	✘	✓
reduce_scatter	✘	✘	✘	✘	✘	✓
all_to_all	✘	✘	✓	?	✘	✓
barrier	✓	✘	✓	?	✘	✓

ML Communication backend APIs



Basic execution model

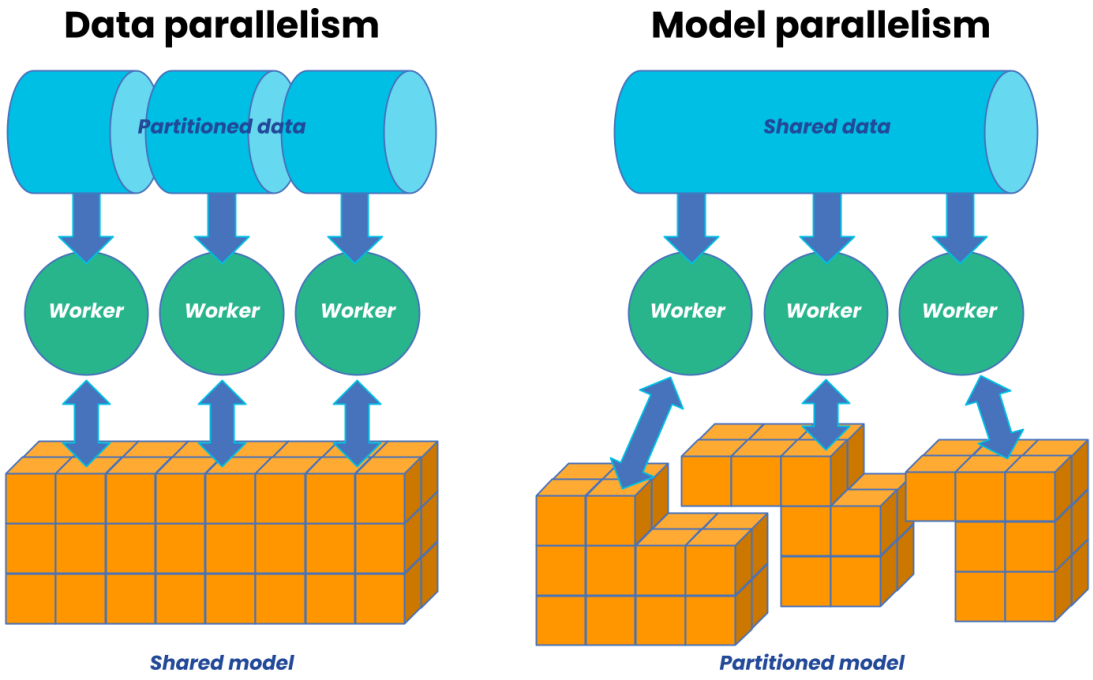


Workers must have sufficient memory to store data, weights, activations & gradients

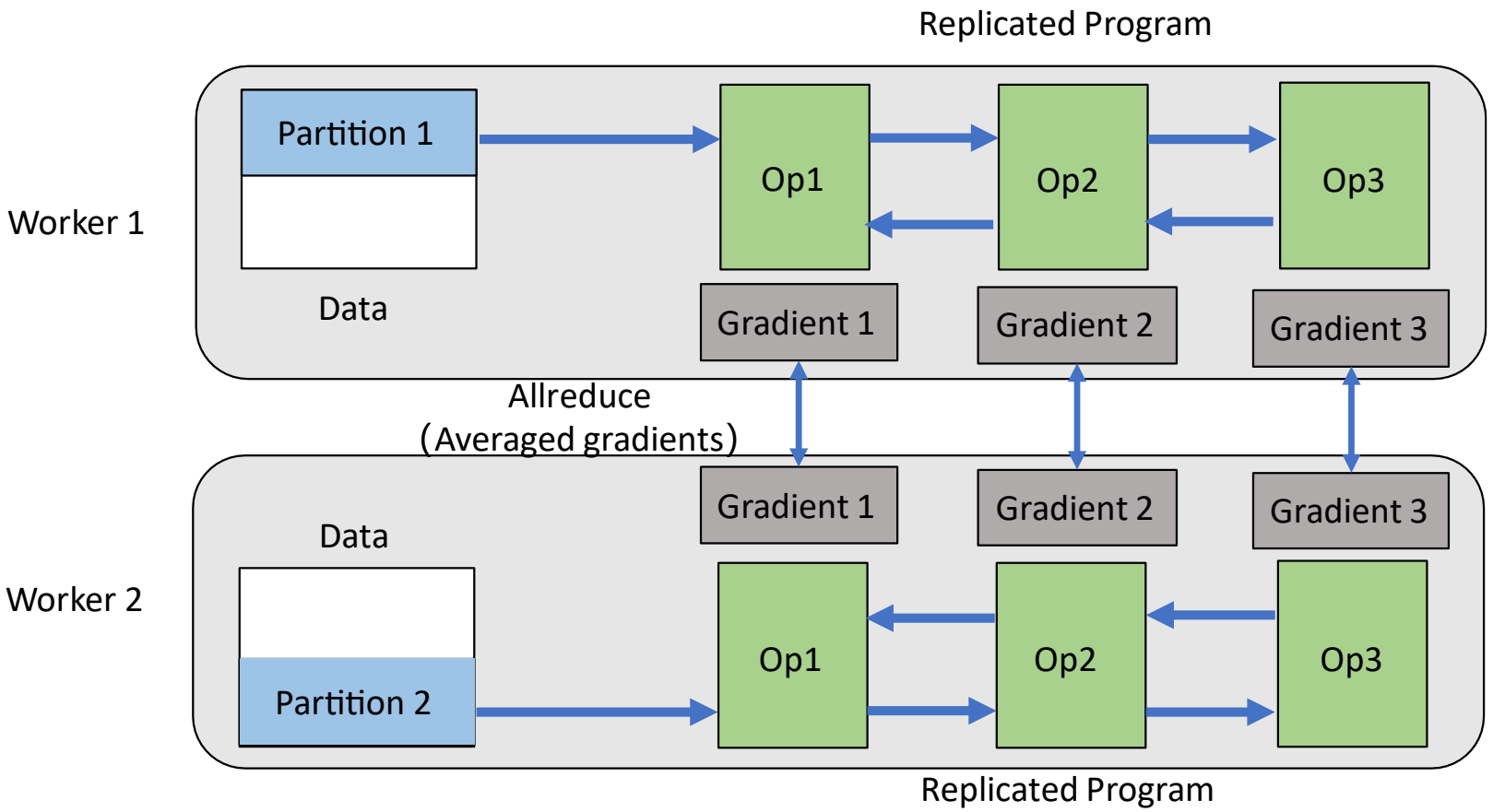
- Otherwise, you get an Out-Of-Memory (OOM) exception

Types of Parallelism in Distributed ML

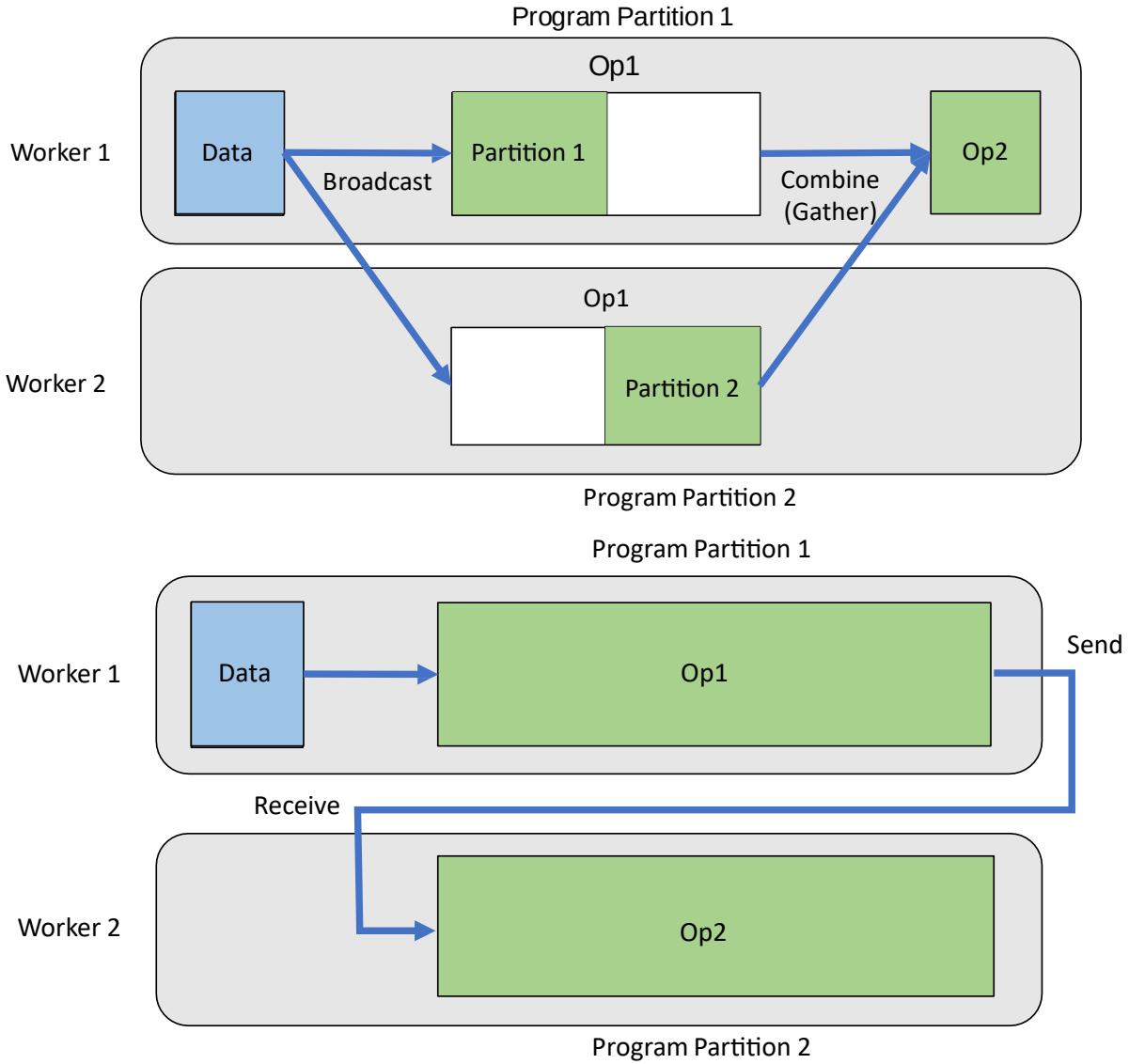
- Data Parallelism ->Input data is partitioned
- Model Parallelism -> Model is partitioned
- Hybrid Parallelism



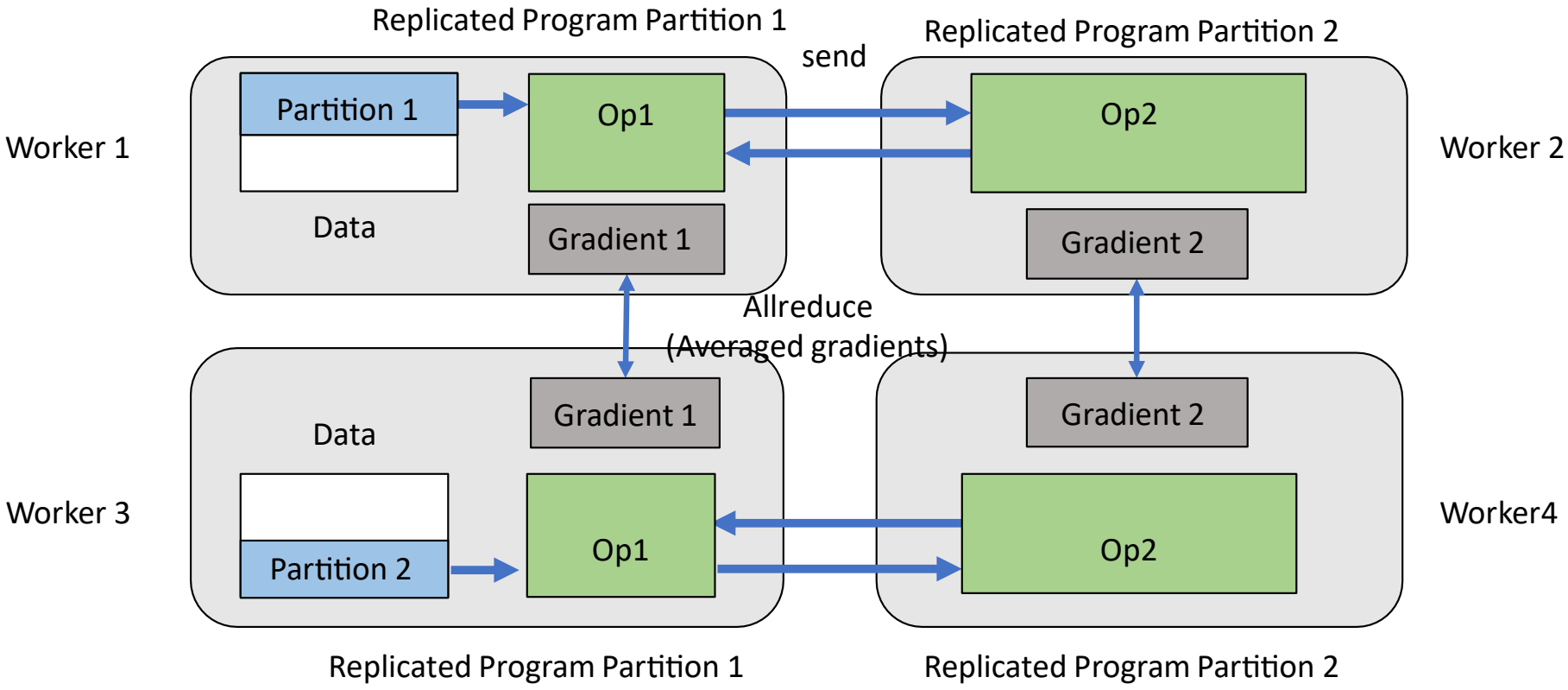
Data Parallelism



Model Parallelism



Hybrid Parallel Training

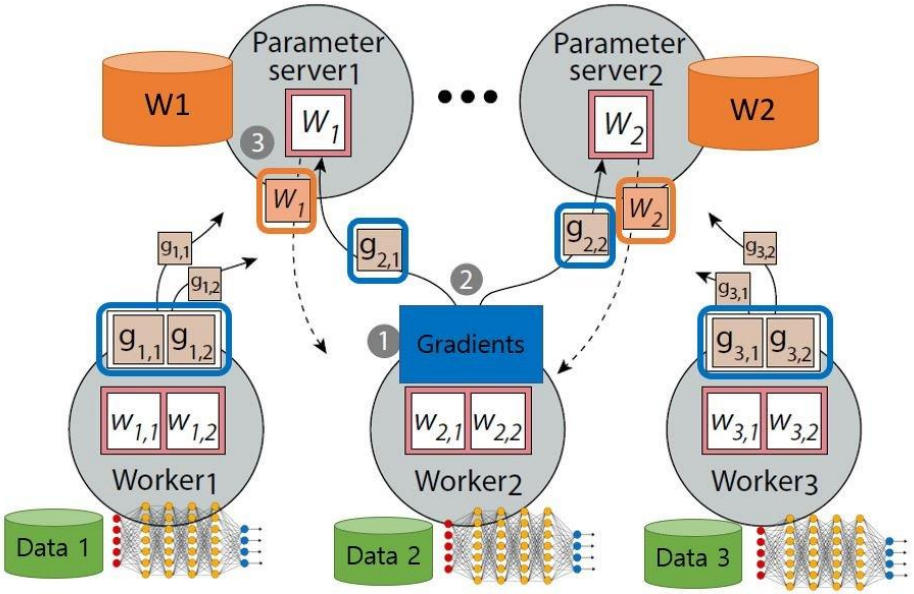


Parameter Server

- Limitation in workers' memory
  - Centralized Parameter Server
  - Decentralized Parameter Server

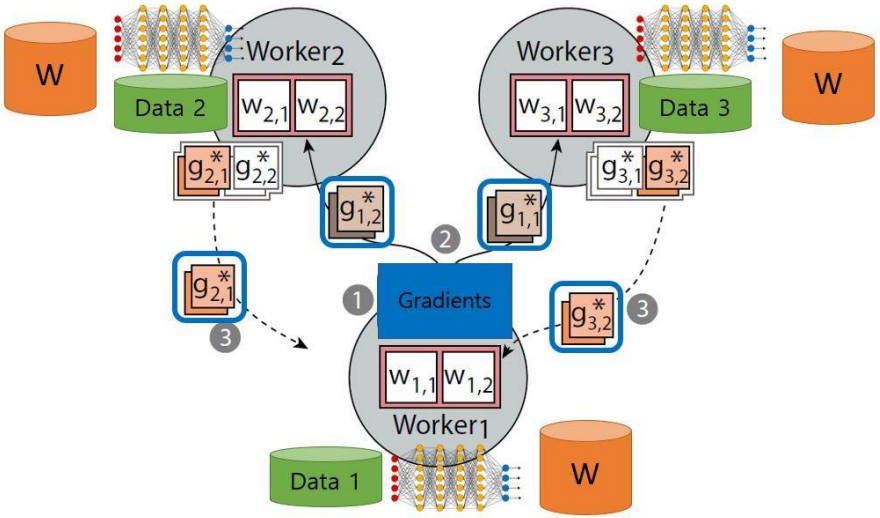
Centralized Parameter Server

1. The worker pulls the latest value of the weights
2. The worker calculates gradients
3. Push gradients to PS
4. PS updates weights



Decentralized Parameter Server

1. The worker pulls the latest value of the weights
2. The worker calculates gradients
3. Push gradients to **other workers**
4. Workers update their weights



Each worker uses part of its memory to keep the parameters

Resources

- Compulsory Reading
  - PyTorch Distributed: Experiences on Accelerating Data Parallel Training https://www.vldb.org/pvldb/vol13/p3005-li.pdf
- Recommended Reading
  - **Ray: A Distributed Framework for Emerging AI Applications**
  - **PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel**
  - **TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches**

QUESTIONS?