슈퍼컴퓨터에서 멀티노드기반 분산딥러닝하기

Soonwook Hwang

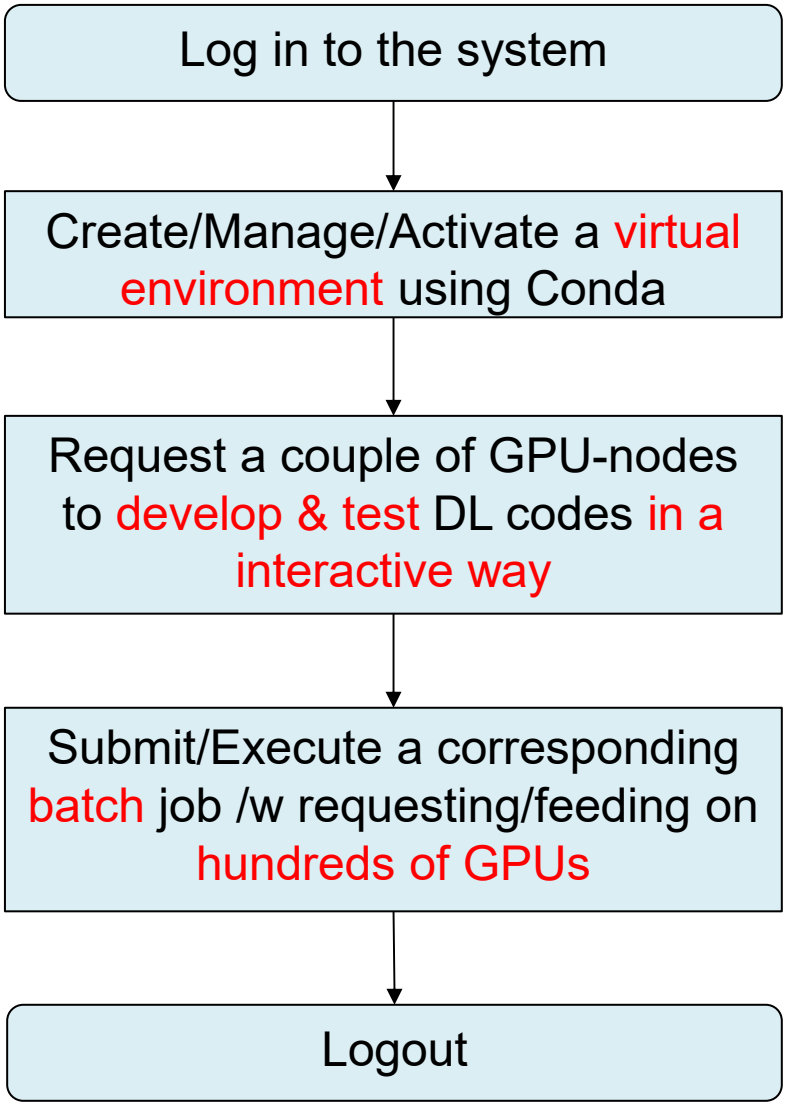
June 18, 2023

Korea Institute of Science and Technology Information

Some Motivational thoughts on large-scale DL pract*ices* on top of national supercomputing facilities

- **KISTI-6 /w ~600PF is coming in 2024**
  - ✓ It is expected that several thousands of GPUs(?) are available
- **Is a large-scale LM (Language Model) training an exclusive(?) task that can be conducted by big tech companies (e.g., Google, Meta, MS) running data center facilities?**
- **Why is it that large-scale LM (LLM) R&D is so hard in Korea?**
  - ✓ Lack of computing resources
  - ✓ Lack of datasets
  - ✓ Lack of skills??
- **What can KISTI do in contributing to large-scale LM R&D in Korea?**
  - ✓ KISTI is uniquely positioned to running a genenal-purpose national supercomputing facilitiy in Korea
- **Is KISTI's supercomputing facility easy to access for users to do large-scal distributed deep learning R&D?**
- **What are the most significant barriers that prevent users from having access to KISTI supercomputing facilities in conducting large-scale distributed training?**
  - ✓ Is it because of the tranditional batch-scheduling based service?
  - ✓ ??

Distributed Training Common Practices/Routines on Supercomputers

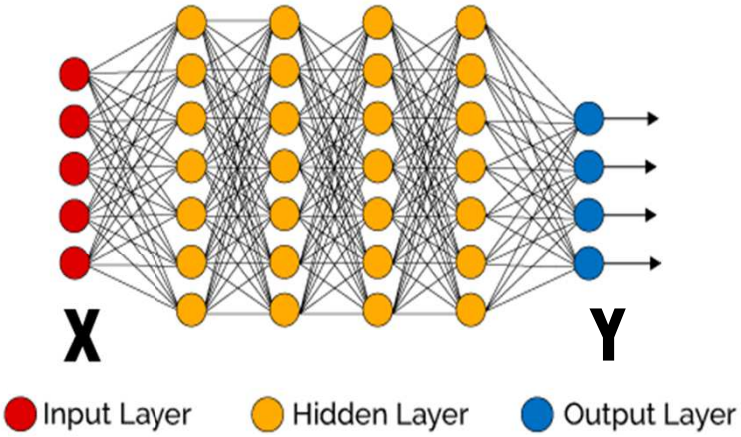


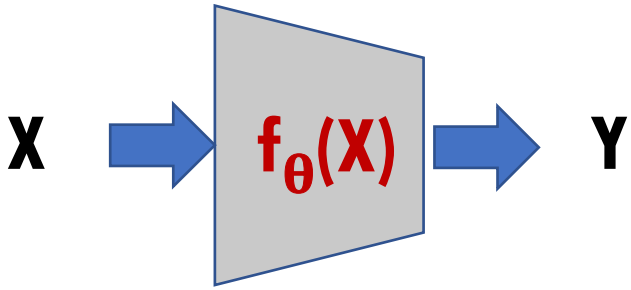
Agenda

- **Why distributed training?**
- **Distributed DL Approaches**
  - Data Parallel
    - Parameter Server(s)
    - Ring AllReduce
  - Model Parallel
    - Pipeline parallel
    - Tensor parallel
  - Multi-dimensional (3D) Parallel
- **KISTI GPU Cluster: Neuron**
- **Hands-on Exercises**
  - Conda Virtual Environment
  - Distributed Data Parallel(DDP) using Horovod
  - DDP using Singularity Container

Deep Learning in a slide

Courtesy of Prof. Sanjeev Arora





θ : parameters

(X, Y) : (Input, Label) pair of Training data

$\ell(\theta, x, y)$ Loss function (how well net output matched true label y on point x) Can be l_2, cross-entropy….

Objective $\text{argmin}_\theta E_i[\ell(\theta, x_i, y_i)]$

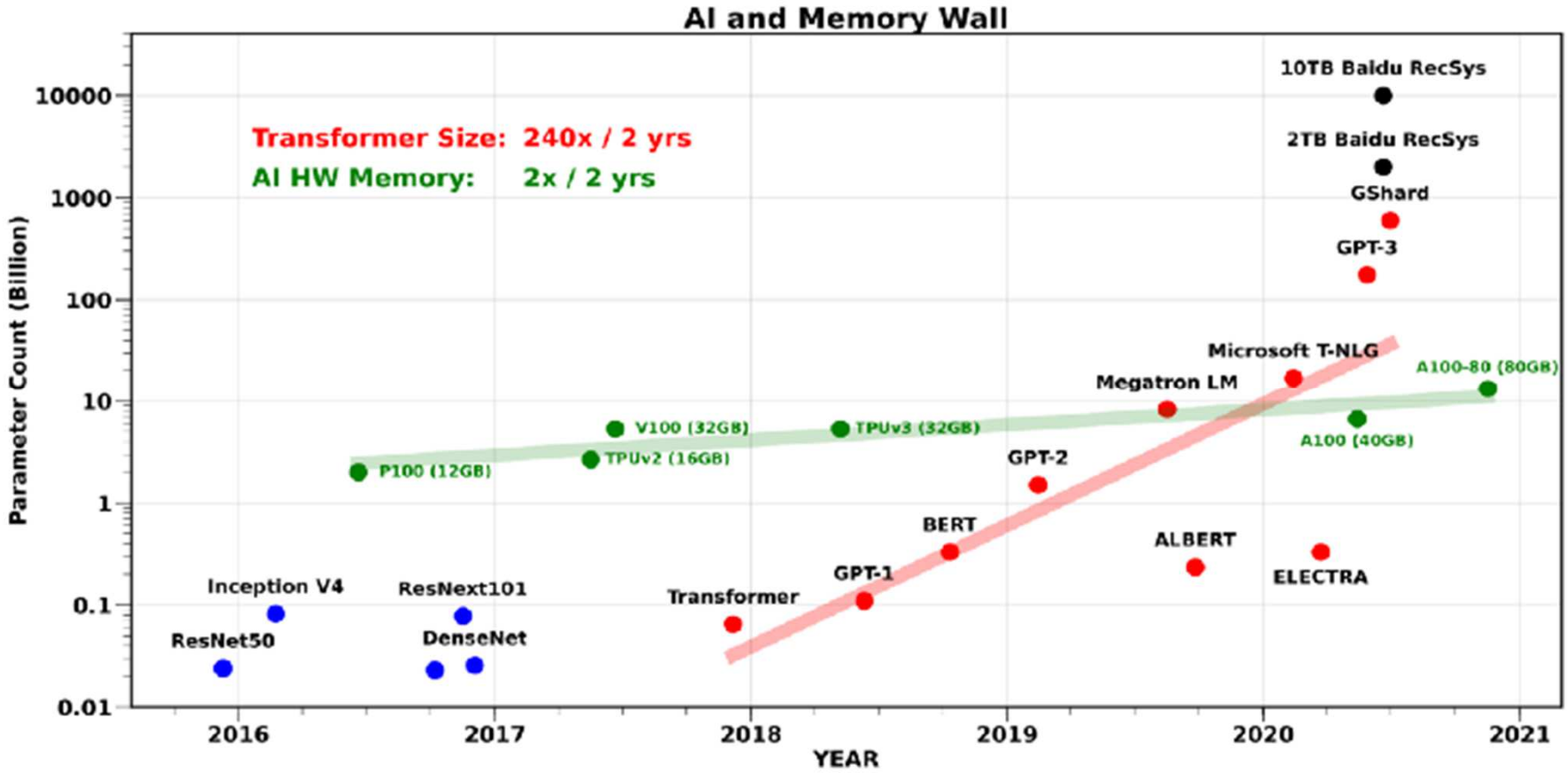
Gradient Descent $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_\theta (E_i[\ell(\theta^{(t)}, x_i, y_i)])$

Stochastic GD: Estimate ∇ via small sample of training data.

Why Distributed Training

Why Distributed Training

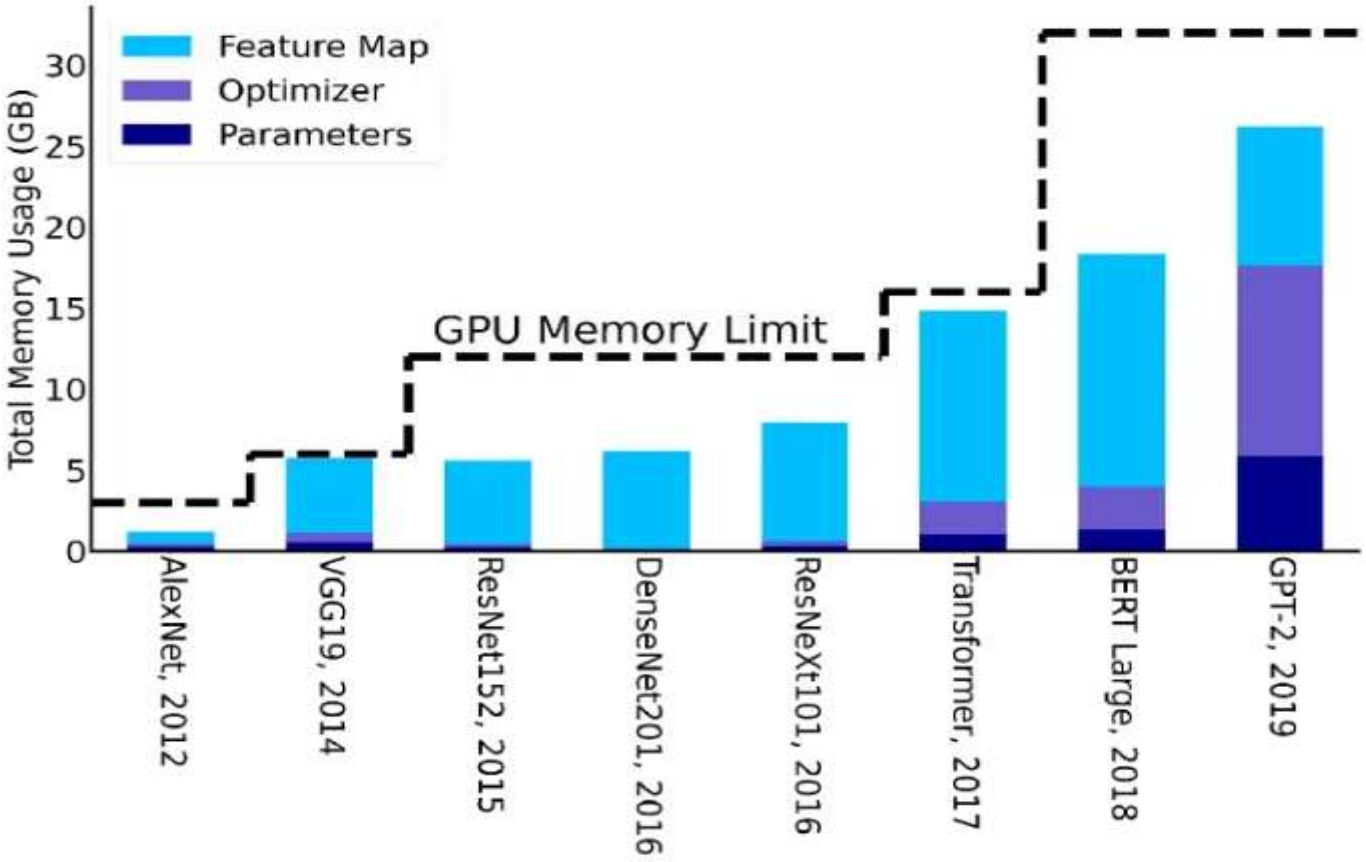
- **Model size is growing too big to fit**
  - GPT-1(‘18): 110M, GPT-2(‘19):1.5B GPT-3(’20):175B



https://medium.com/riselab/ai- and- memory- wall- 2cb4265cb0b8

AI and Memory Wall

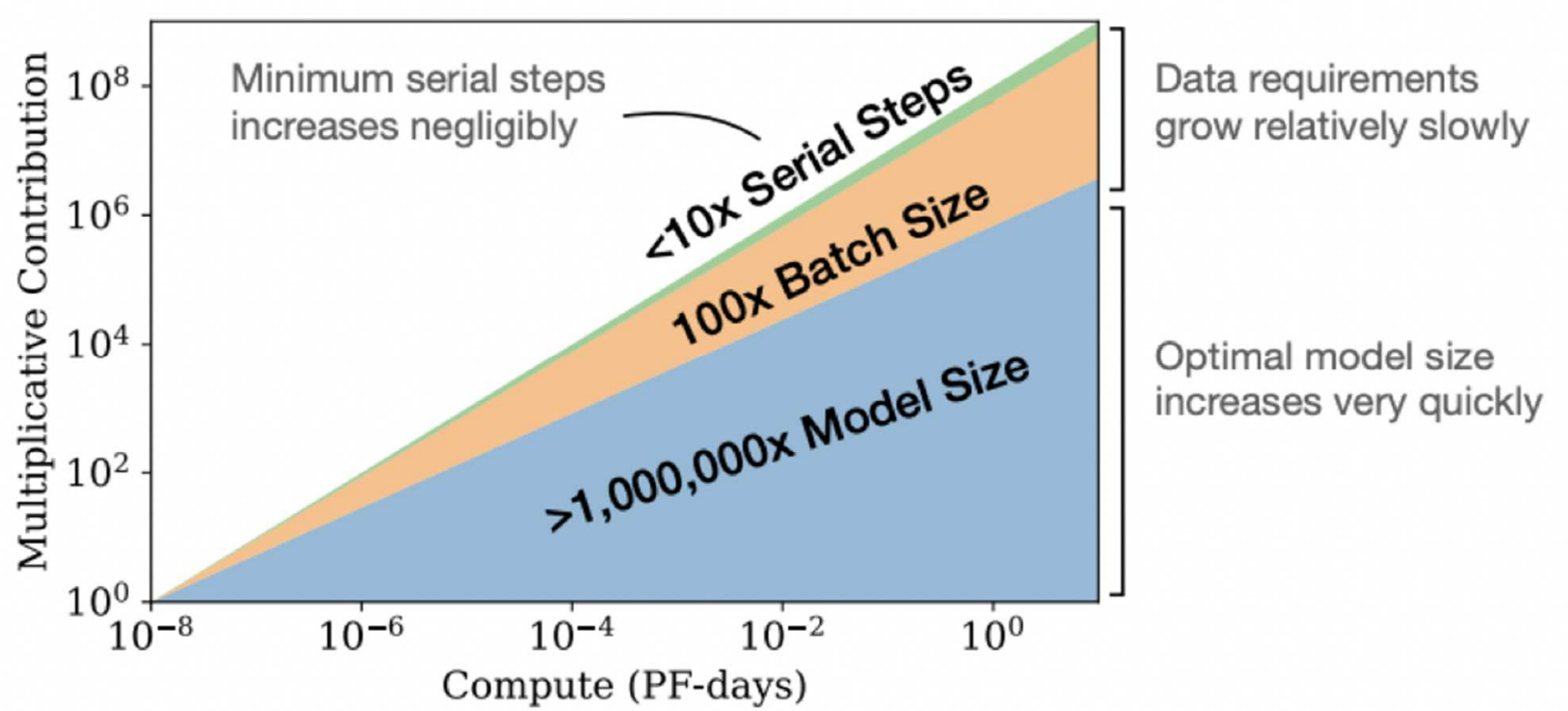
- **The amount of memory required to train different DL models**



https://medium.com/riselab/ai- and- memory- wall- 2cb4265cb0b8

Scaling Laws

- **Larger models are significantly more sample-efficient**
  - ✓ optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.



Scaling Laws for Neural Language Models
https://arxiv.org/pdf/2001.08361.pdf

Why Distributed Training

- **With More compute power and memory using multiple devices (GPU/TPU/CPU)**
  - Enable training large model
  - Speed up model training and shorten training time
  - Shorter training time allows you to do more experiments to reach your modeling goal

Scaling with Distributed Training

- **(Tensorflow) Training ResNet50 with ImageNet data on Neuron (A100)**
  - /apps/applications/singularity_images/imagenet (~150G)
    - train #: ~1,280,000, validation #: 50,000, test #: 100,000 images
  - GPU #8: 10.1 min/epoch, GPU #4: 17.6 min, GPU #1: 134.7min

```
[gpu37] $ mpirun –np 8 python tf_keras_imagenet_resnet50.py
Epoch 1/90
5004/5004 [==============================] - 604s 118ms/step - loss: 6.5324 -
accuracy: 0.0560 - top_k_categorical_accuracy: 0.1529
```

```
[gpu37] $ mpirun –np 4 python tf_keras_imagenet_resnet50.py
Epoch 1/90
10009/10009 [==============================] - 1054s 104ms/step - loss:
6.3358 - accuracy: 0.0678 - top_k_categorical_accuracy: 0.1794
```

```
[gpu37] $ mpirun –np 1 python tf_keras_imagenet_resnet50.py
Epoch 1/90
40037/40037 [==============================] - 8081s 202ms/step - loss:
6.0712 - accuracy: 0.0817 - top_k_categorical_accuracy: 0.2053
```

Scaling with Distributed Training

- **(Pytorch) Training ResNet50 with ImageNet data on Neuron**
  - /apps/applications/singularity_images/imagenet (~150G)
    - train #: ~1,280,000, validation #: 50,000, test #: 100,000 images
  - GPU #8: 15.04 min/epoch, GPU #4: 28.19 min, GPU #1: 134.7min

```
[gpu37] $ mpirun –np 8 python pytorch_imagenet_resnet50.py
Train Epoch     #1: 100%|                    | 5005/5005 [15:26<00:00,  5.40it/s, loss=5.63,
accuracy=5.33]
```

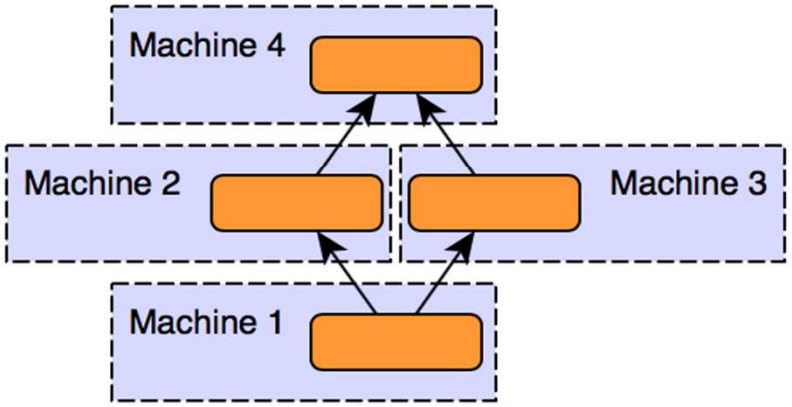
```
[gpu37] $ mpirun –np 4 python pytorch_imagenet_resnet50.py
Train Epoch     #1: 100%|                    | 10010/10010 [28:19<00:00,  5.89it/s,
loss=5.53, accuracy=6.03]
```

```
[gpu37] $ python tf_keras_imagenet_resnet50.py
Epoch 1/90
40037/40037 [==============================] - 8081s 202ms/step - loss:
6.0712 - accuracy: 0.0817 - top_k_categorical_accuracy: 0.2053
```

Distributed DL Approaches

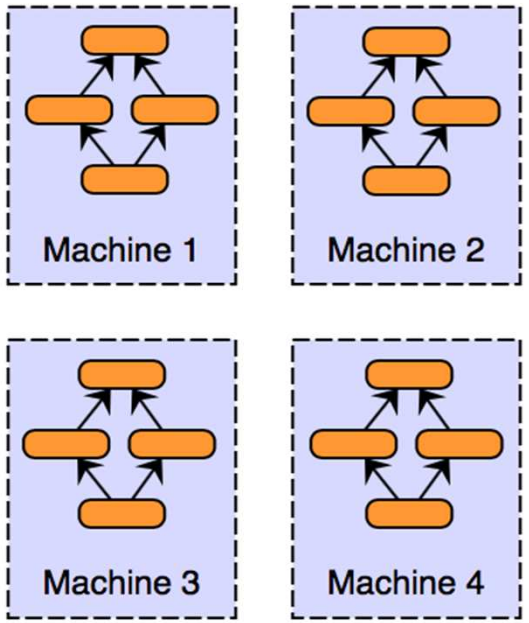
Distributed DL approaches

Model Parallelism



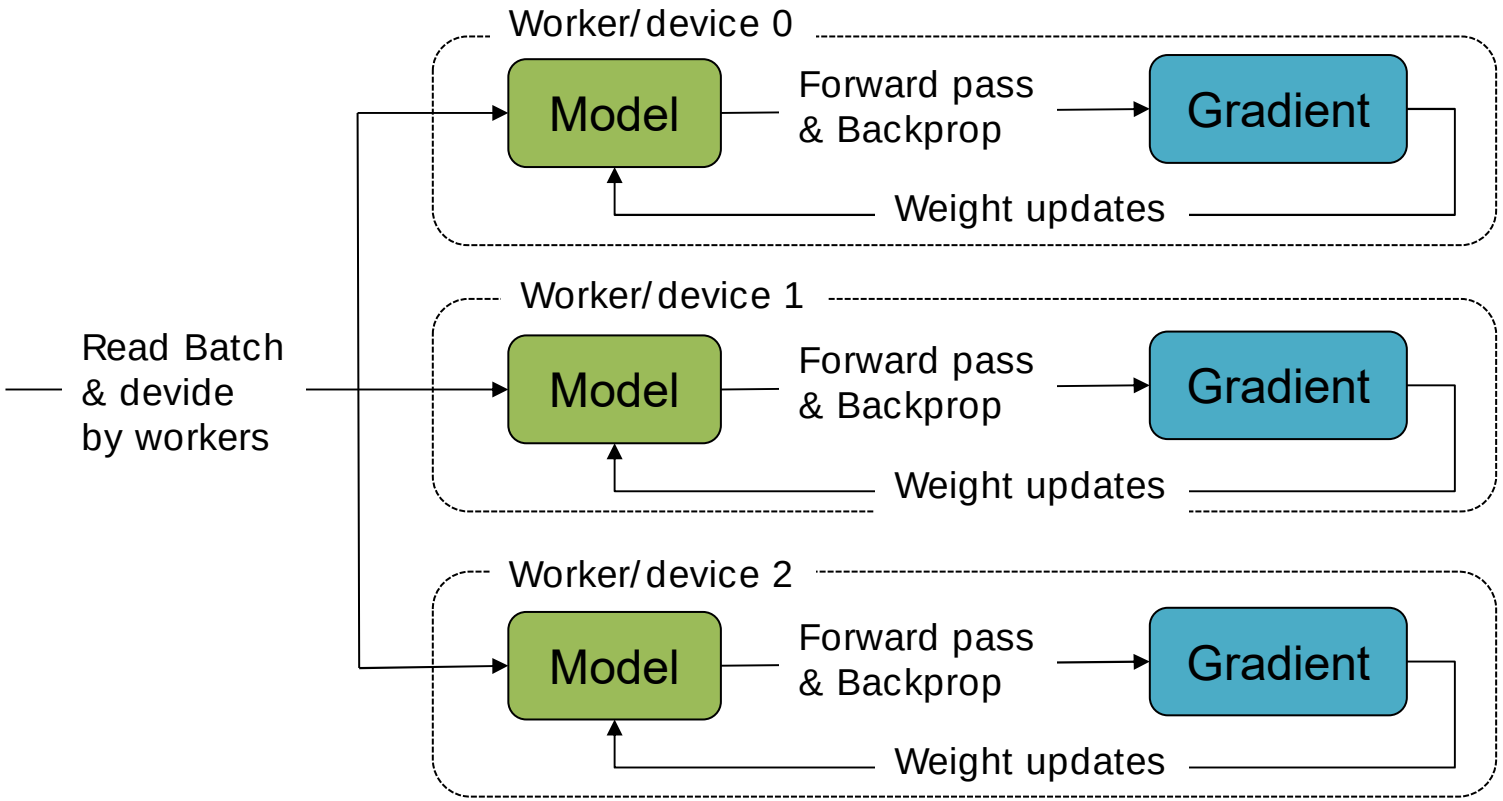
- Different parts of **model** running on multiple GPUs
- Model is too large, which cannot fit in a single device

Data Parallelism



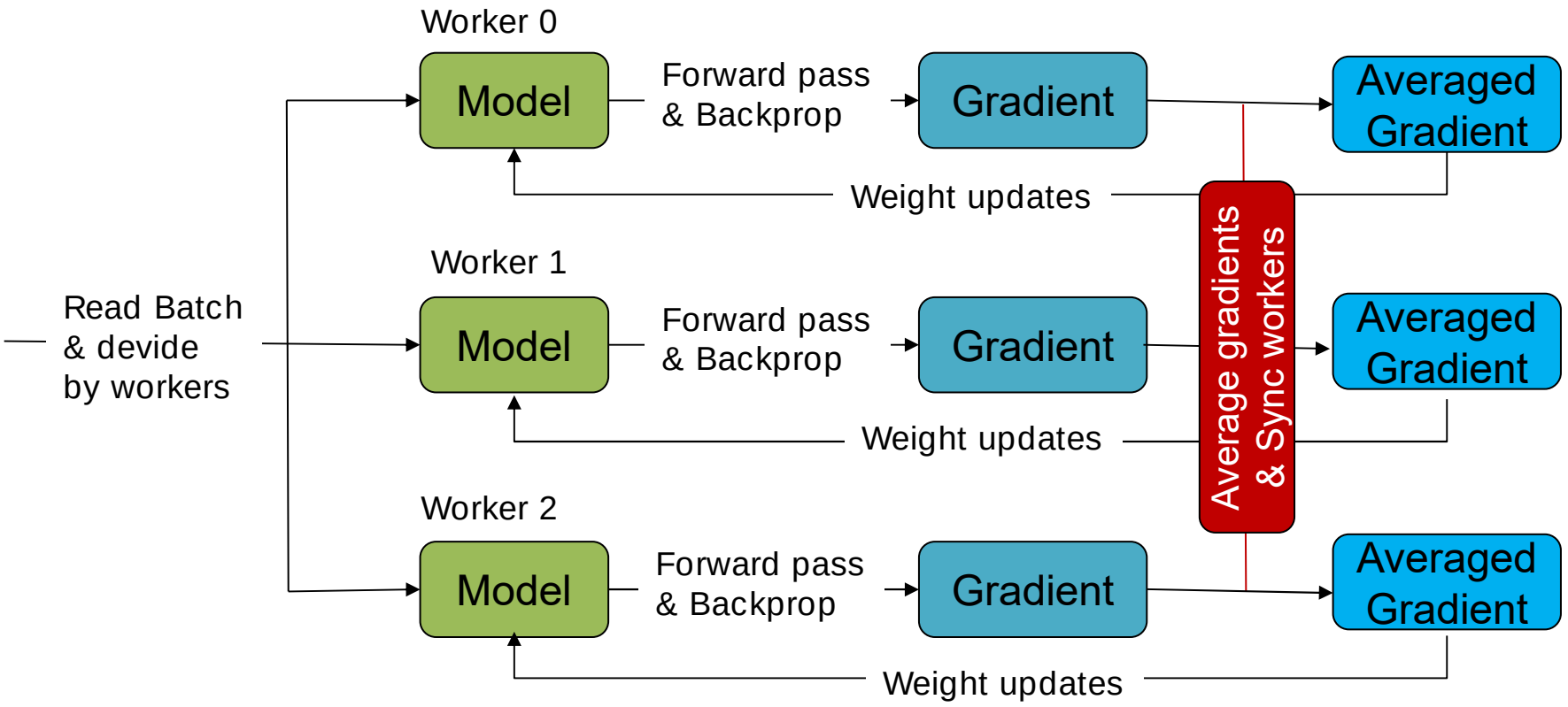
- Different parts of **data** running on multiple GPUs
- Data is too large, which need to be processed in parallel

Data Parallel w/o Sync



- **How to divide/distribute batches to workers**
- **When/How to synchoronize workers**
- **How to maintain consistency in states**

Data Parallel



How to synchronize model parameters

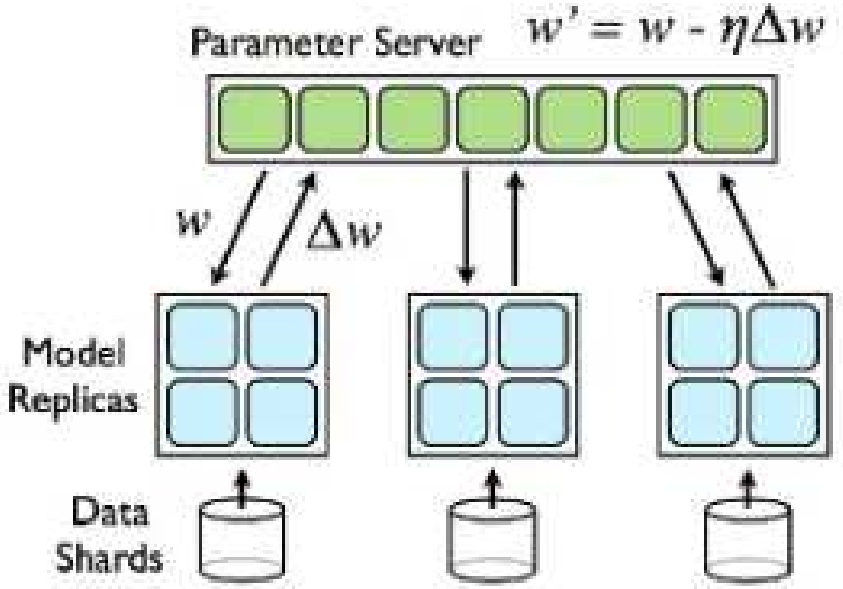
- **Parameter Server (centralized)**
  - Synchronous
  - Asynchronous
- **Sync AllReduce (decentralized)**

Parameter Server (PS)

1. **Split the training data into shards** and assign a model replica to each data shard

2. For each model replica, fetch the parameters from the centralized **sharded parameter server**

3. Gradients are computed per model and pushed back to the parameter server



Each data shard stores a subset of the complete training data

Asynchronous vs. Synchronous PS

Downpour SGD:
Online Asynchronous Stochastic Gradient Descent

Sandblaset L- BFGS:
Batch Distributed Parameter Storage and Manupulation

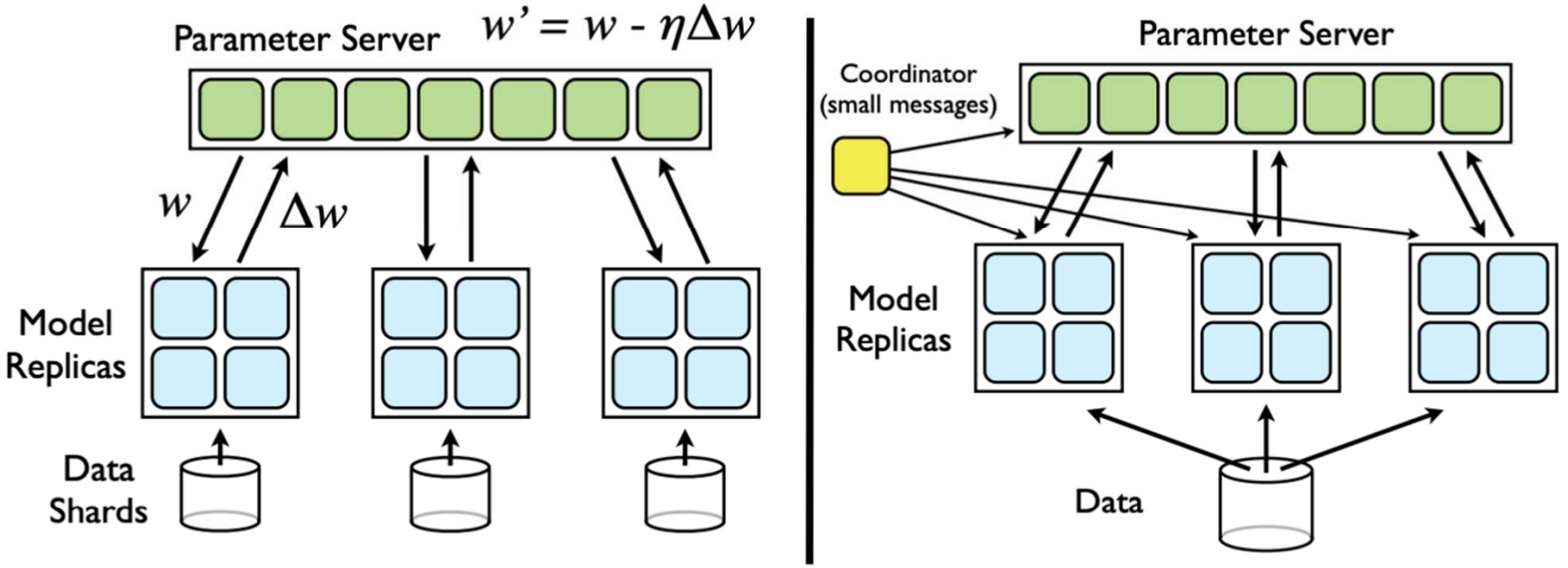
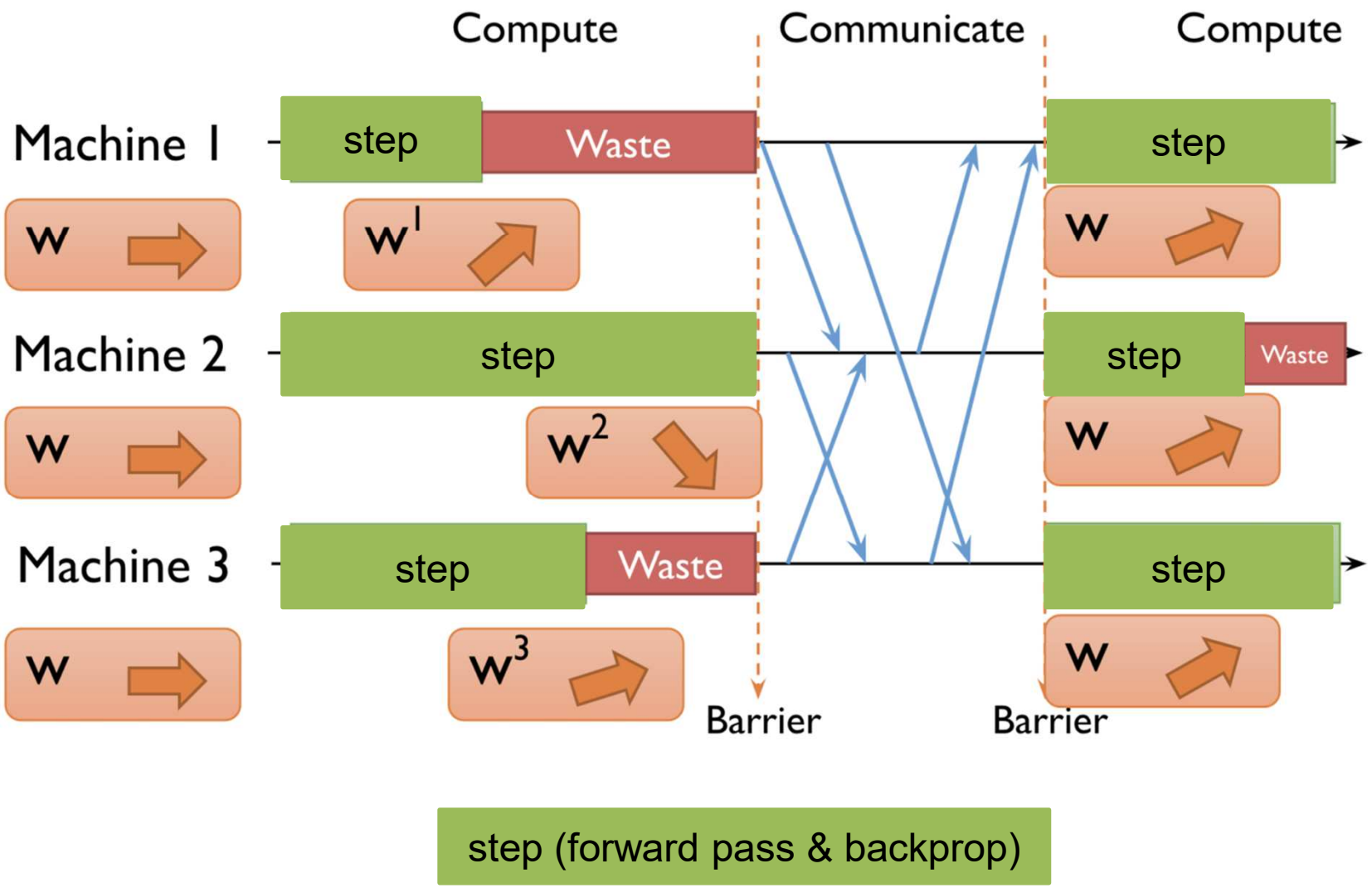


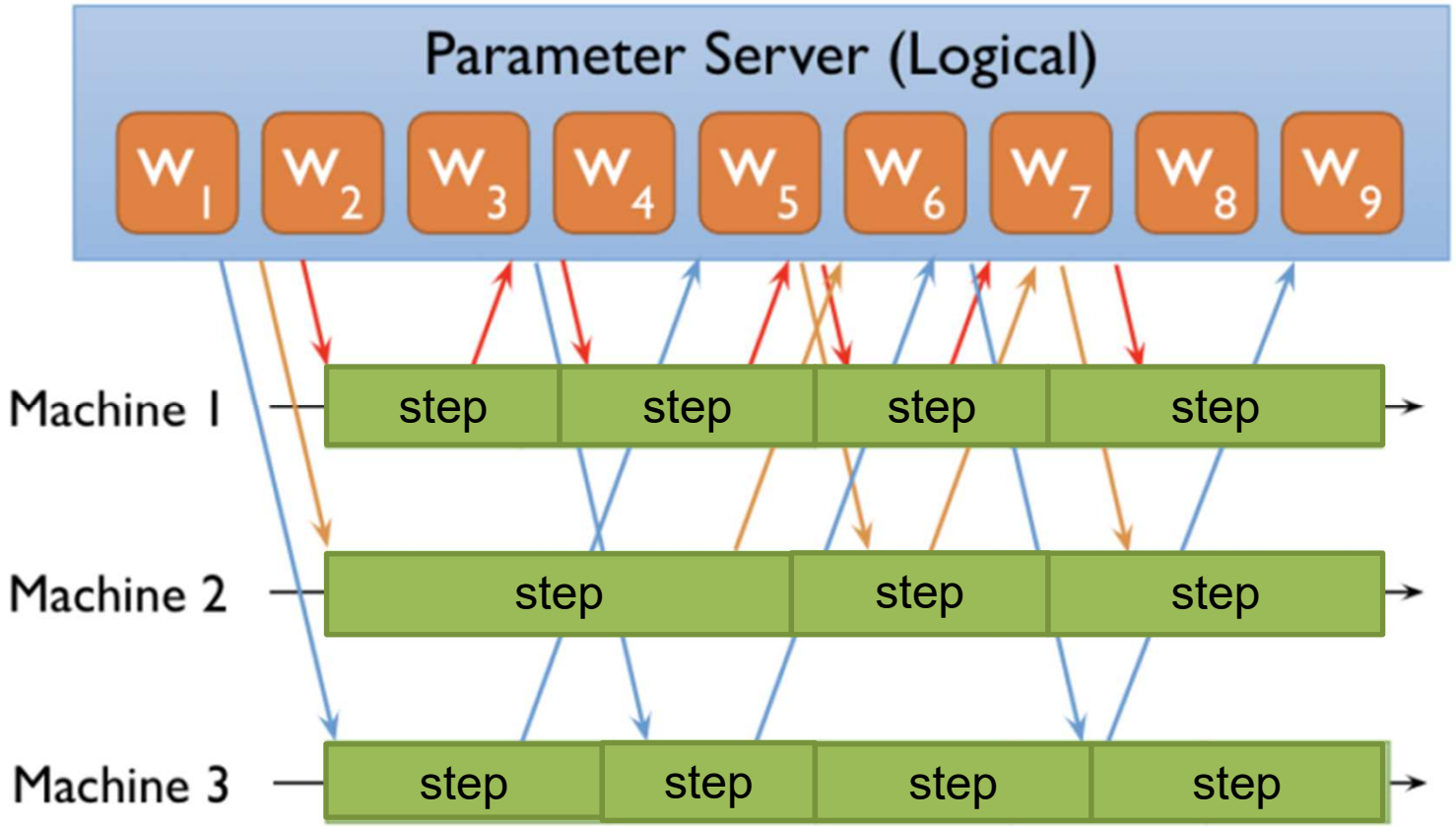
Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters w and push gradients Δw to the parameter server. Right: Sandblaster L-BFGS. A single 'coordinator' sends small messages to replicas and the parameter server to orchestrate batch optimization.

Figure from Large Scale Distributed Deep Networks
https://static.googleusercontent.com/media/research.google.com/ko//archive/large_deep_networks_nips2012.pdf

Synchronous PS

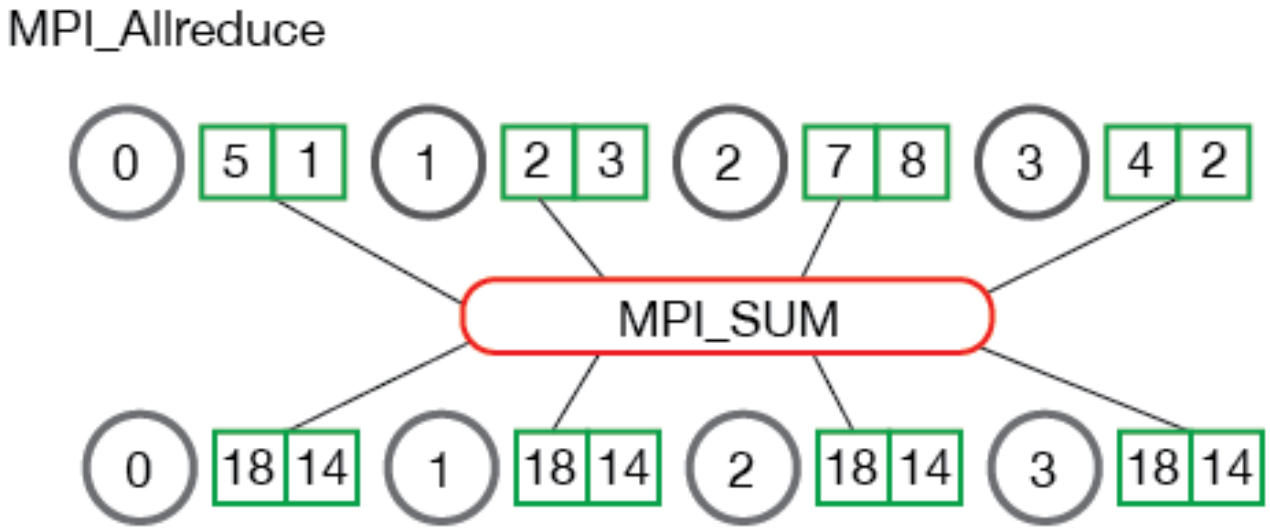


Asynchronous PS



AllReduce operation

- a collective communication operation that reduce a set of arrays on distributed workers to a single array that is then re-distributed back to each workers



Ring AllReduce

- **Synchronized w/o parameter server(s)**

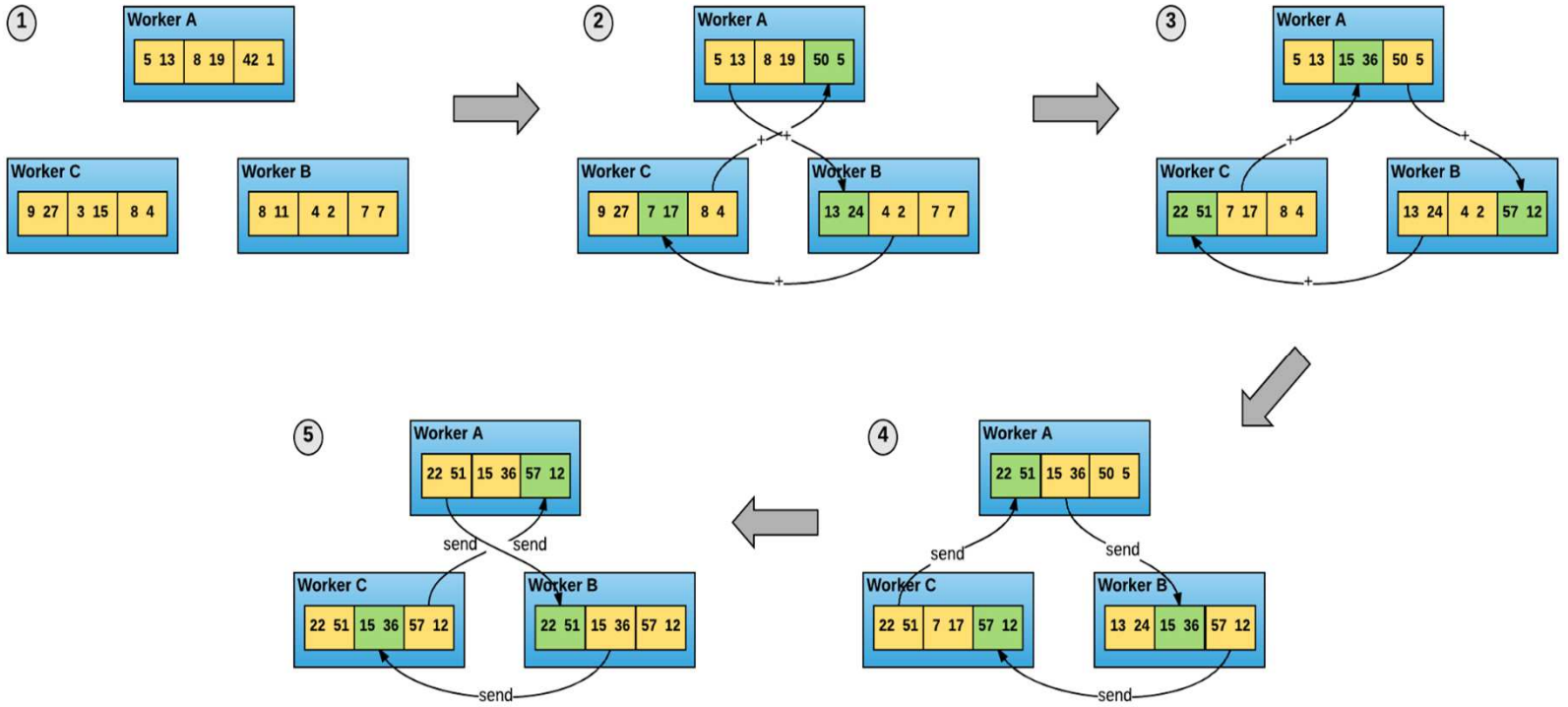
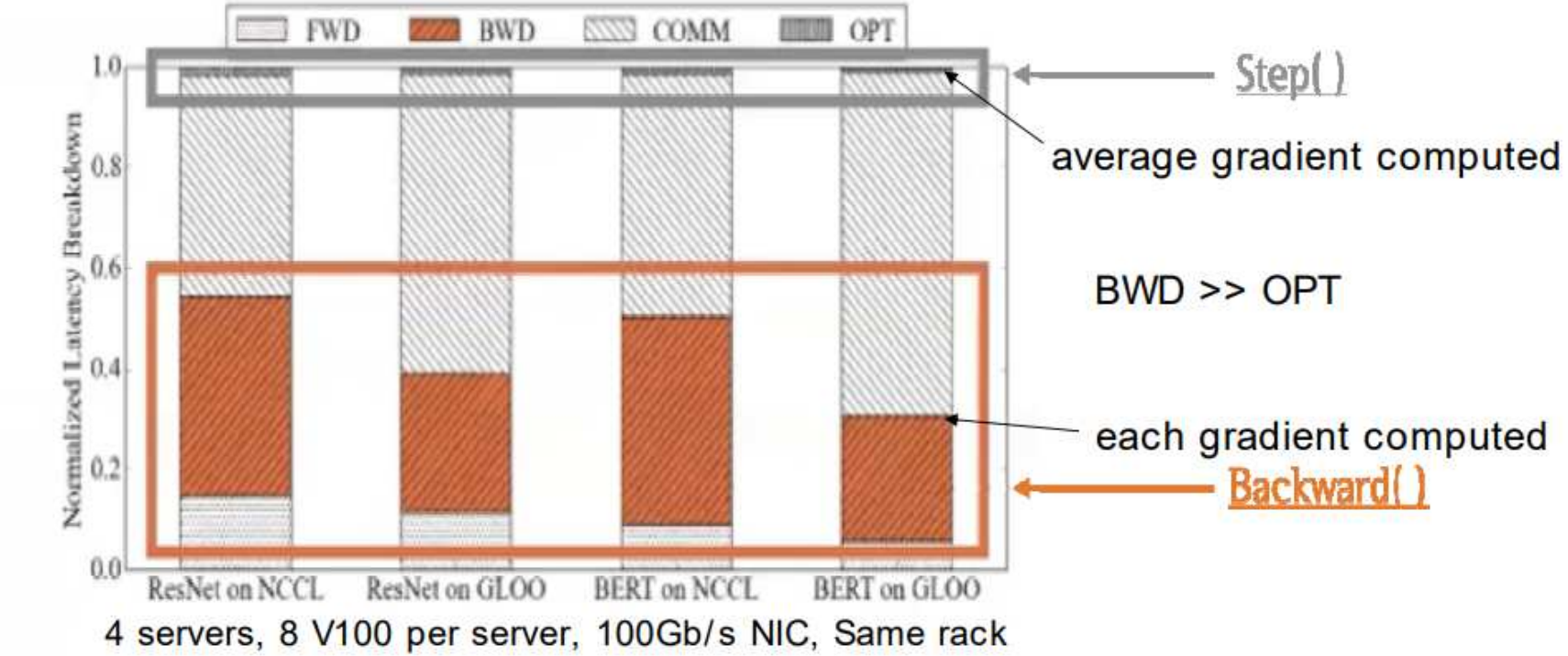


Figure 4: The ring-allreduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.

Horovod: fast and easy distributed deep learning in TensorFlow

https://arxiv.org/pdf/1802.05799.pdf

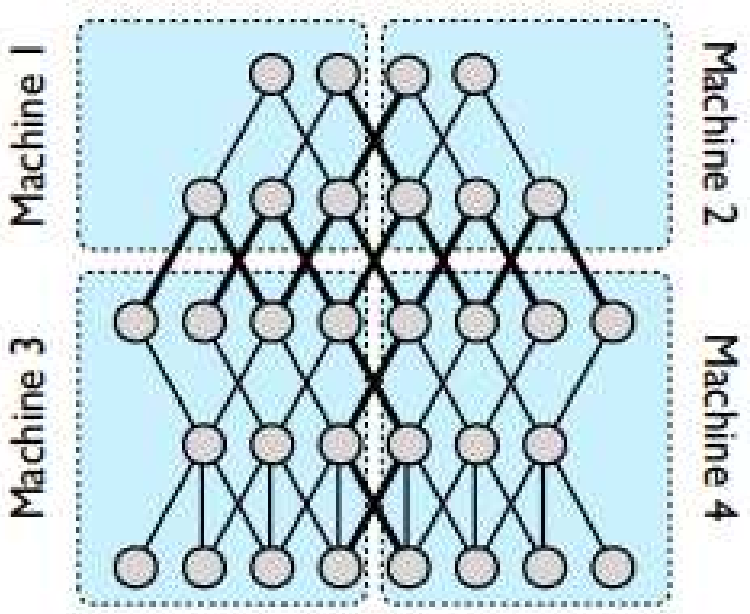
Latency Breakdown of Data Parallelism



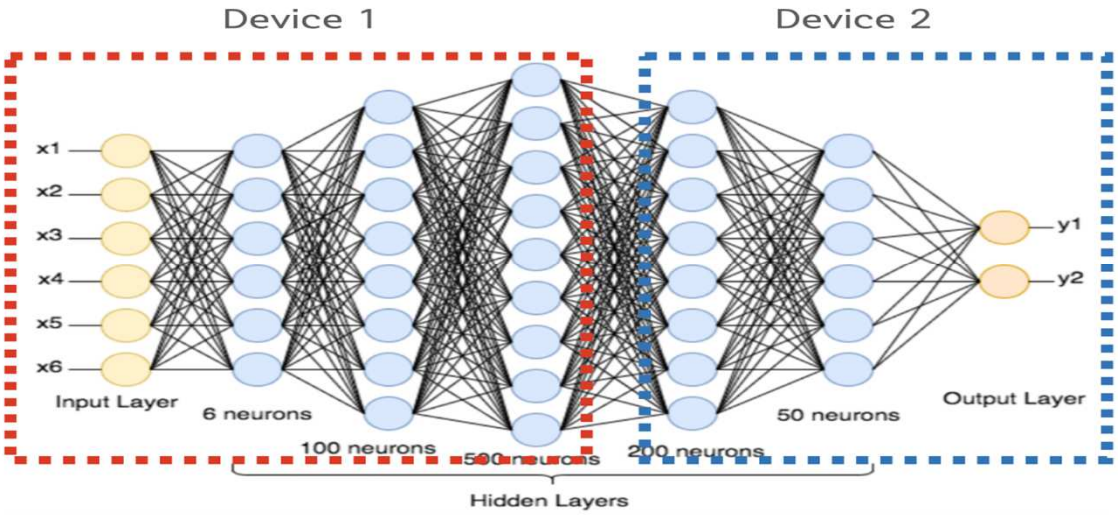
- **COMM is very expensive**
- **What if averaging model parameters after OPT (optimizer step) rather than averaging gradients?**
- **Is there a way to reduce latency and make it fast?**

Model Parallel

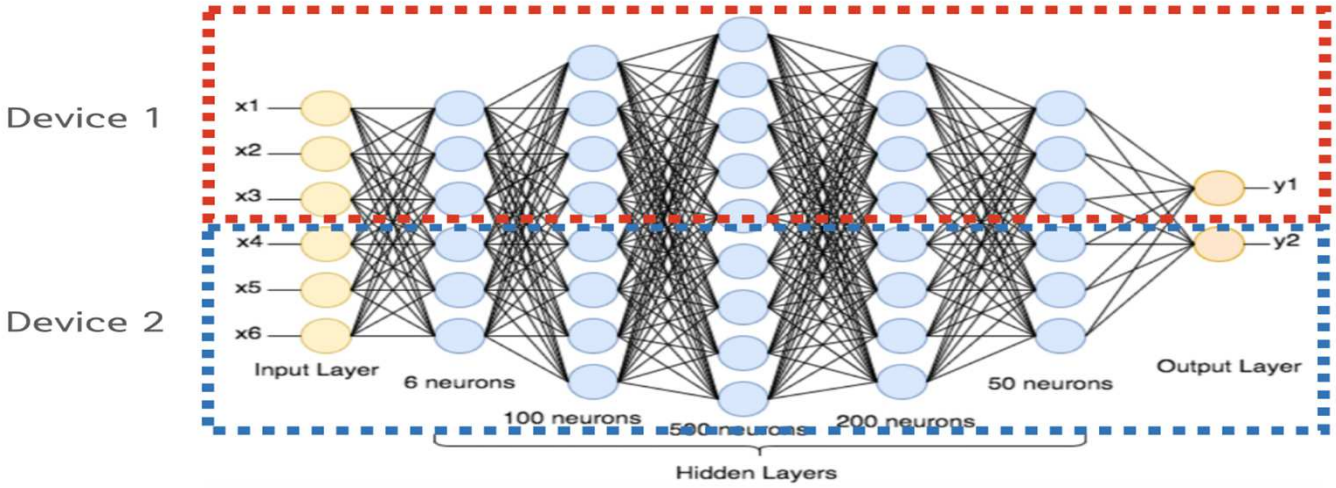
- **Inter-layer Model Parallel**
  - pipeline parallel
- **Intra-layer Model Parallel**
  - Tensor parallel
- **Multi-dimensional Parallel**
  - 3D parallel



Inter/Intra-layer Model Parallel

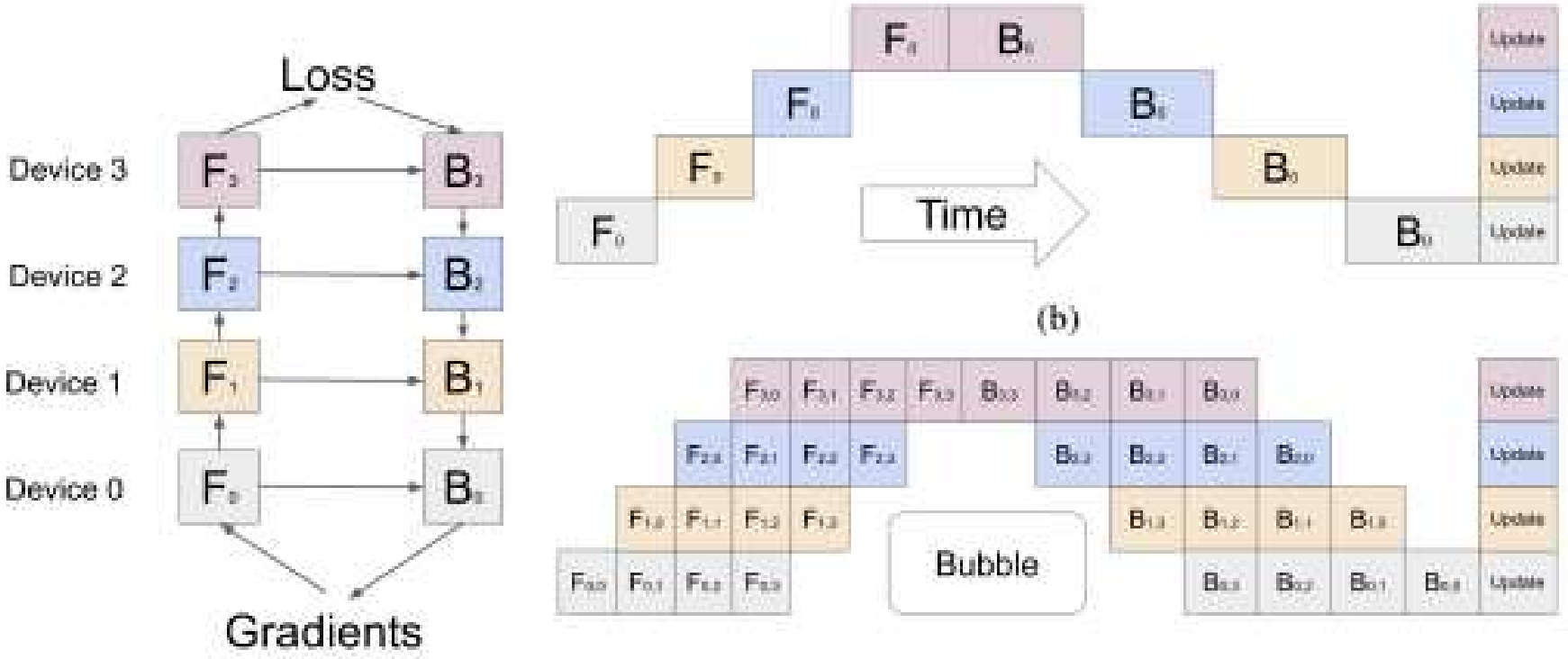


Inter- layer model parallel



Intra- layer model paralle

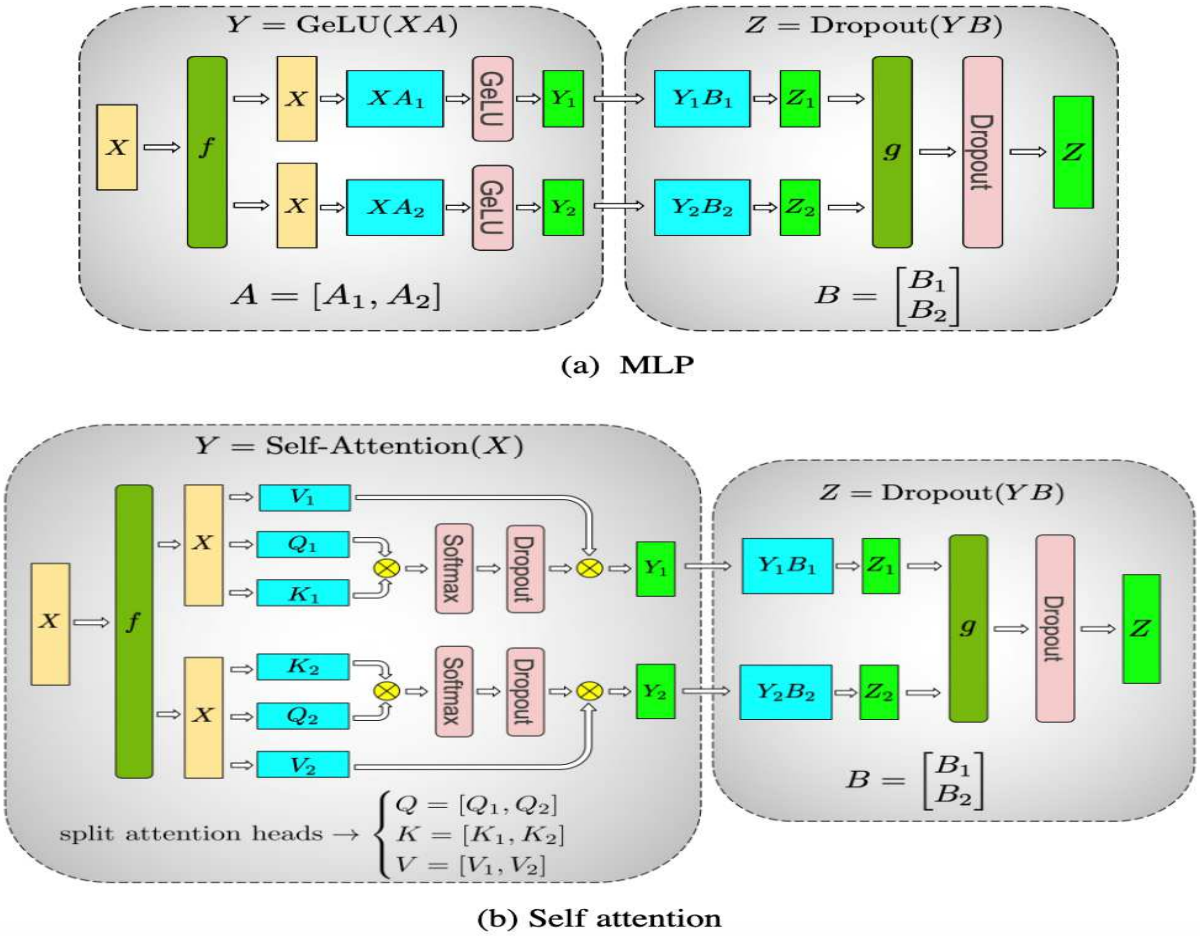
Pipeline Model Parallel



GPipe: Easy Scaling with Micro- Batch Pipeline Parallelism
https://arxiv.org/pdf/1811.06965.pdf

PipeDream: Fast and Efficient Pipeline Parallel DNN Training
https://arxiv.org/pdf/1806.03377.pdf

Tensor Parallel

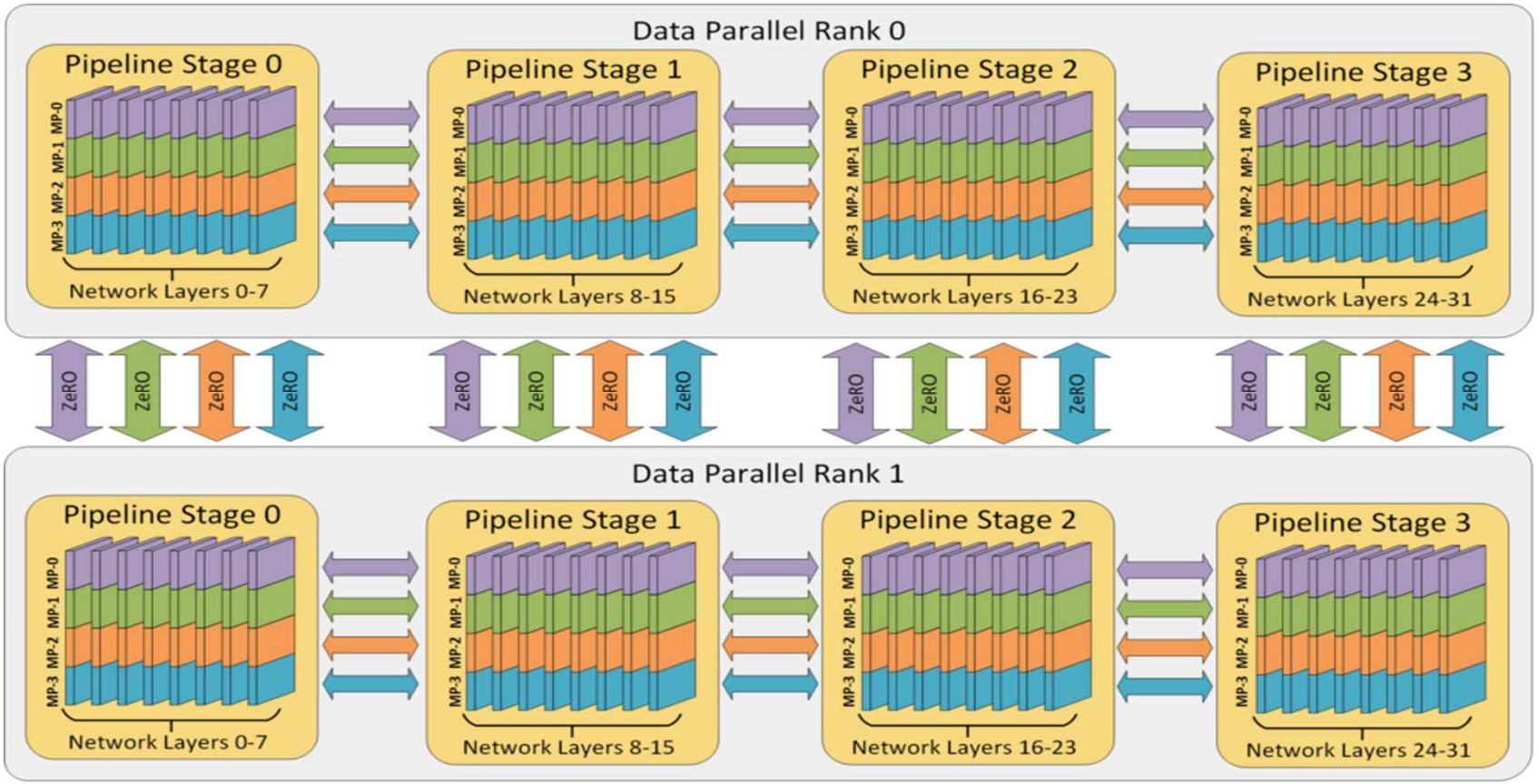


Megatron- LM: Training Multi- Billion Parameter Language Models Using Model Parallelism

https://arxiv.org/pdf/1909.08053.pdf

3D Parallel

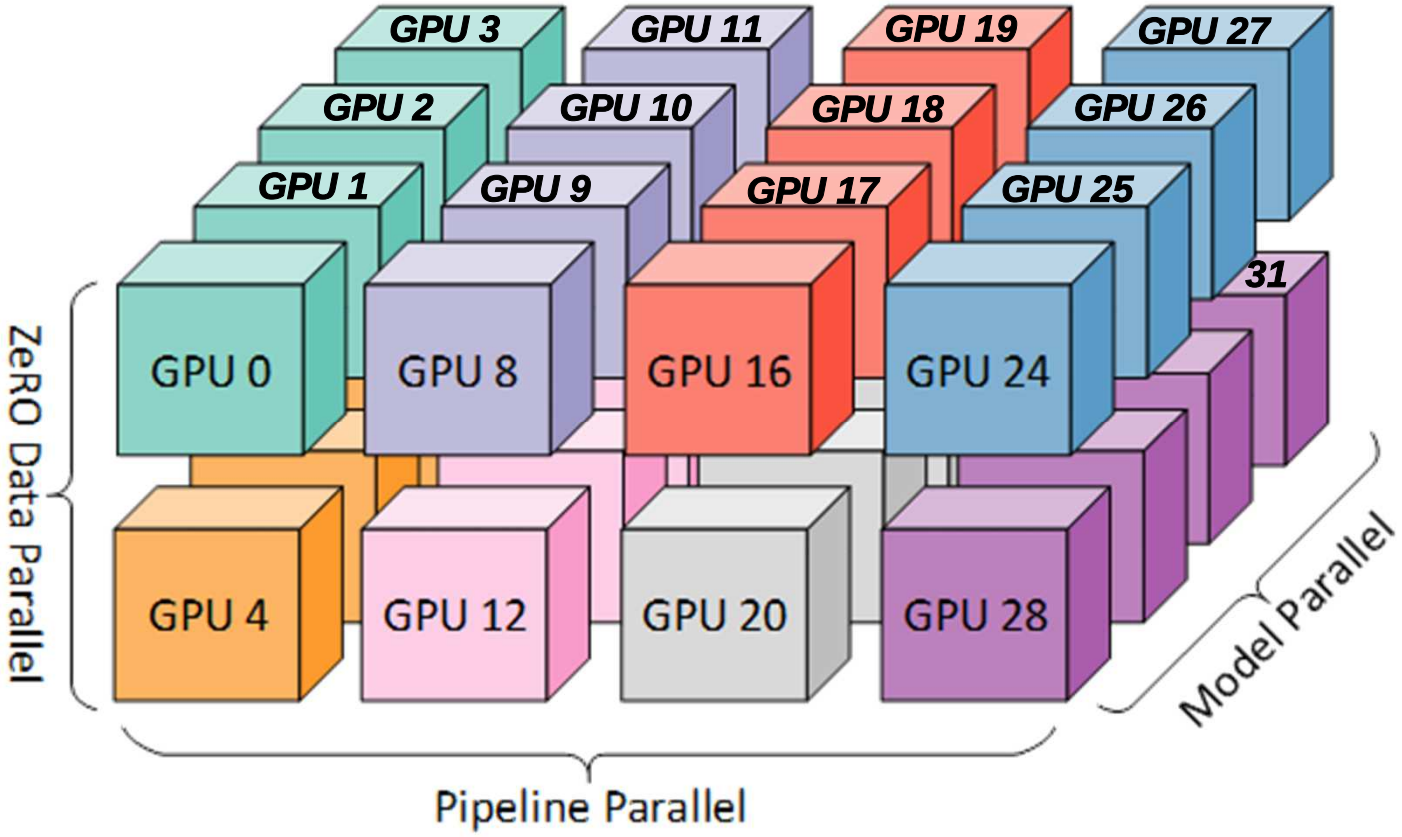
- **Data Parallel + Pipeline Parallel + Tensor Parallel**
  - 32 workers (2DP x 4PP x 4TP)



DeepSpeed: Extreme- scale model training for everyone

https://www.microsoft.com/en- us/research/blog/deepspeed- extreme- scale- model- training- for- everyone

3D Parallel Example (2 x 4 x 4)



Megatron-Turing NLG (530B)

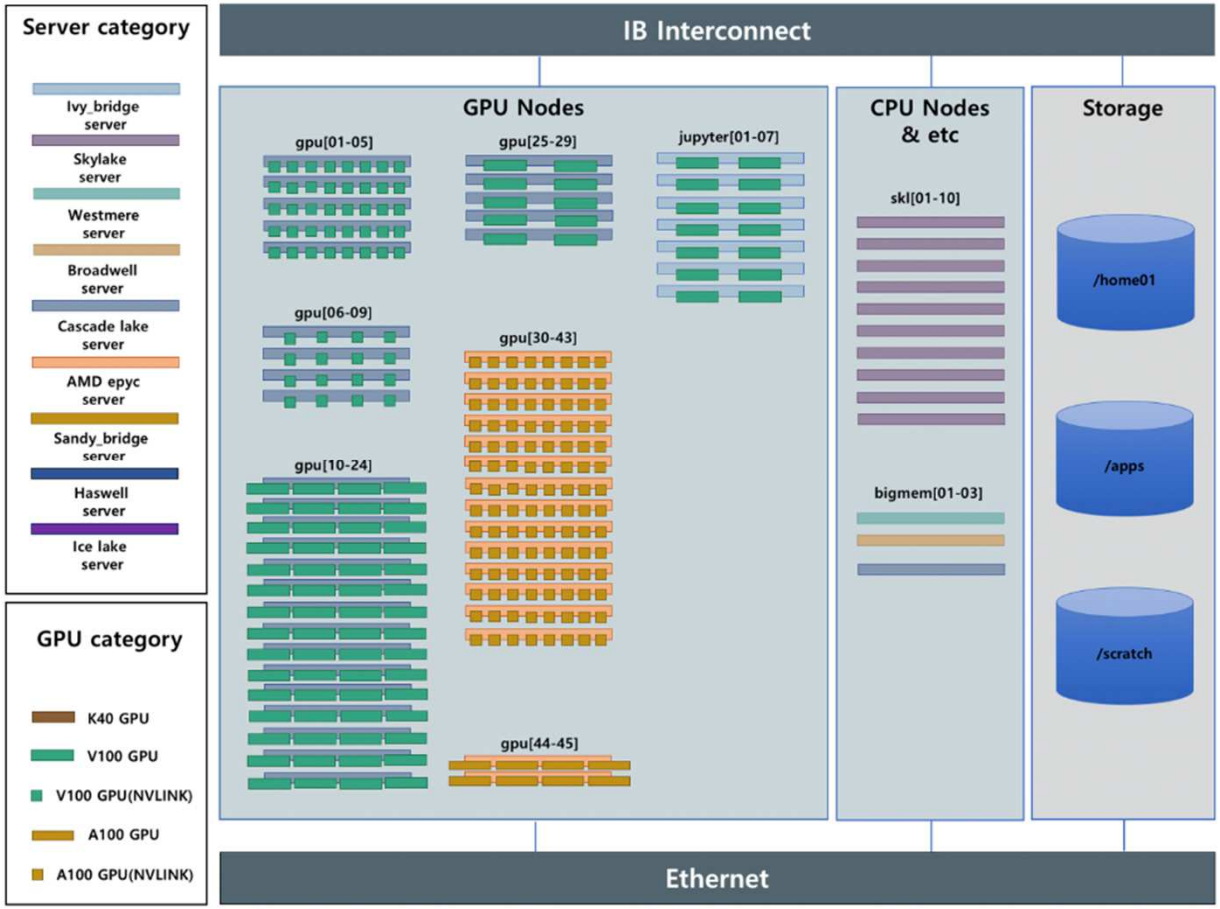
MT-NLG

- **Based on NVIDIA Megatron + MS DeepSpeed**
- **Trained on the NVIDIA DGX SuperPOD-based Selene supercomputer**
  - powered by 560 DGX A100 servers networked with HDR InfiniBand
- **3D parallel system**
  - 8-way tensor slicing within node
  - 35-way pipeline parallelism across nodes
  - 2-way data parallelism
- **Transformer-based 530 bilion parameters**
  - # of layers: 105
  - # of hidden dimensions: 20480
  - # of attention heads: 128
- **Sequence length: 2048**
- **Global Batch size: 1920**

NVIDIA Technical Blog: Using DeepSpeed and Megatron to Train Megatron- Turing NLG 530B
https://developer.nvidia.com/blog/using- deepspeed- and- megatron- to- train- megatron- turing- nlg- 530b- the- worlds- largest- and- most- powerful- generative- language- model/

KISTI GPU Cluster: Neuron

KISTI GPU Cluster: Neuron



year	#node	#GPU	#Flops
2018	44	53	312.8TF
2019	63	99	698.8TF
2020	78	163	1.24PF
2021	59	200	2.34PF
2022	**65**	**260**	3.53PF

Slurm Queues on Neuron

Queue Name (CPU_GPU_GPU#)	#Node	#Total CPU Core	#Job Submission Limit per User	#GPU allocation Limit per User	
cas_v100nv_8	5	160	2	40	V100 (NVlink) 8ea
cas_v100nv_4	4	160	2	16	V100 (NVlink) 4ea
cas_v100_4	15	600	4	40	V100 4ea
cas_v100_2	5	160	2	10	V100 2ea
amd_a100nv_8	14	868	4	64	A100 (Nvlink) 8ea
amd_a100_4	2	128	1	8	A100 4ea
amd_a100_2	2	128	1	4	A100 2ea
skl	10	360	2	-	
bigmem	3	120	1	-	

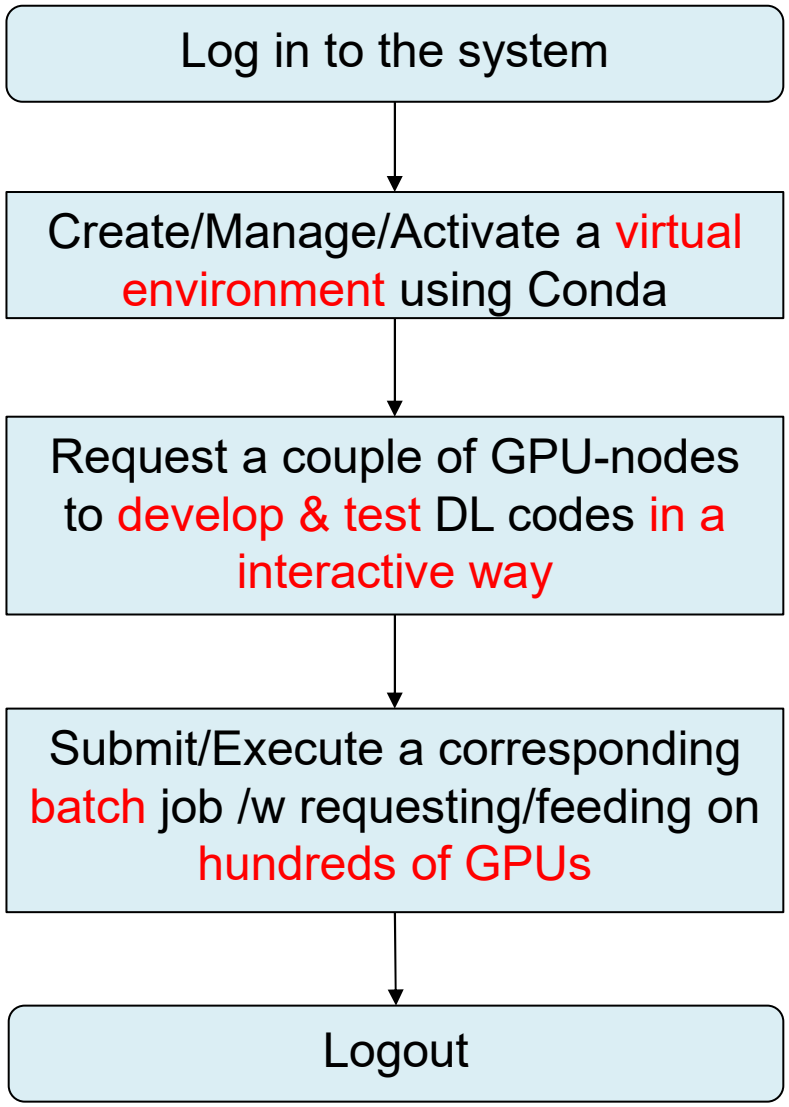
Slurm Queues & available GPUs on Neuron

```
$ sinfo
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
jupyter         up 2-00:00:00      3    mix jupyter[02-04]
jupyter         up 2-00:00:00      1   idle jupyter01
cas_v100nv_8    up 2-00:00:00      1    mix gpu01
cas_v100nv_8    up 2-00:00:00      4  alloc gpu[02-05]
cas_v100nv_4    up 2-00:00:00      1    mix gpu09
cas_v100nv_4    up 2-00:00:00      2  alloc gpu[07-08]
cas_v100_4      up 2-00:00:00      2    mix gpu[13,17]
cas_v100_4      up 2-00:00:00     10  alloc gpu[10-12,18-24]
cas_v100_4      up 2-00:00:00      2   idle gpu[14-15]
cas_v100_2      up 2-00:00:00      1    mix gpu25
cas_v100_2      up 2-00:00:00      1  alloc gpu26
amd_a100nv_8    up 2-00:00:00      2    mix gpu[36-37]
amd_a100nv_8    up 2-00:00:00      6  alloc gpu[30,32-33,39-41]
amd_a100nv_8    up 2-00:00:00      1   idle gpu42
amd_a100_4      up 2-00:00:00      1    mix gpu44
amd_a100_4      up 2-00:00:00      1  alloc gpu45
skl             up 2-00:00:00      8   idle skl[01-06,08-09]
bigmem          up 2-00:00:00      2   idle bigmem[01-02]
exclusive       up   infinite      1    mix gpu06
scidebert       up   infinite      1    mix gpu35
scidebert       up   infinite      1  alloc gpu34
new_service     up   infinite      4  down* bigmem03,jupyter[05-06],skl07
maintenance     up   infinite      6   idle gpu[16,29,31,38,43],jupyter07
```

- **Slurm Quick Start User Guide**
  - https://slurm.schedmd.com/quickstart.html
- **KISTI User Guide**
  - https://www.ksc.re.kr/gsjw/jcs/hd

Distributed Training Practice on Supercomputer

Distributed Training Practices on Neuron




```
## create/activate a virtual environment
$ conda create -n pt_env python=3.7
$ conda activate pt_env
$ (pt_env) conda install pytorch
$ (pt_env) python train.py
```

```
## request 2 nodes for interactive job
$ salloc --nodes=2 --time=8:00:00 --
gres=gpu:4 # available GPU-nodes allocated

## run& test ML/DL codes interactively
$ (pt_env) srun -n 8 python train_ddp.py
```

```
## submit a batch job script requesting 20
nodes with 8 GPUs each node
$ (pt_env) sbatch train_ddp_script.sh

## monitor/check the job status
$ (pt_env) squeue
```

Conda Virtual Environments

Conda Virtual Environment

- **Download & Install Conda on /scratch/userID directory**

```
## Miniconda
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ chmod 755 Miniconda3-latest-Linux-x86_64.sh
$ ./Miniconda3-latest-Linux-x86_64.sh
```

  - ✓ Type your Conda installation directory to "/scratch/userID/miniconda3"
    - ✓ Conda default installation directory: $HOME/miniconda3
  - ✓ Type Conda init: "yes", which will add conda init scripts to ~/.bashrc

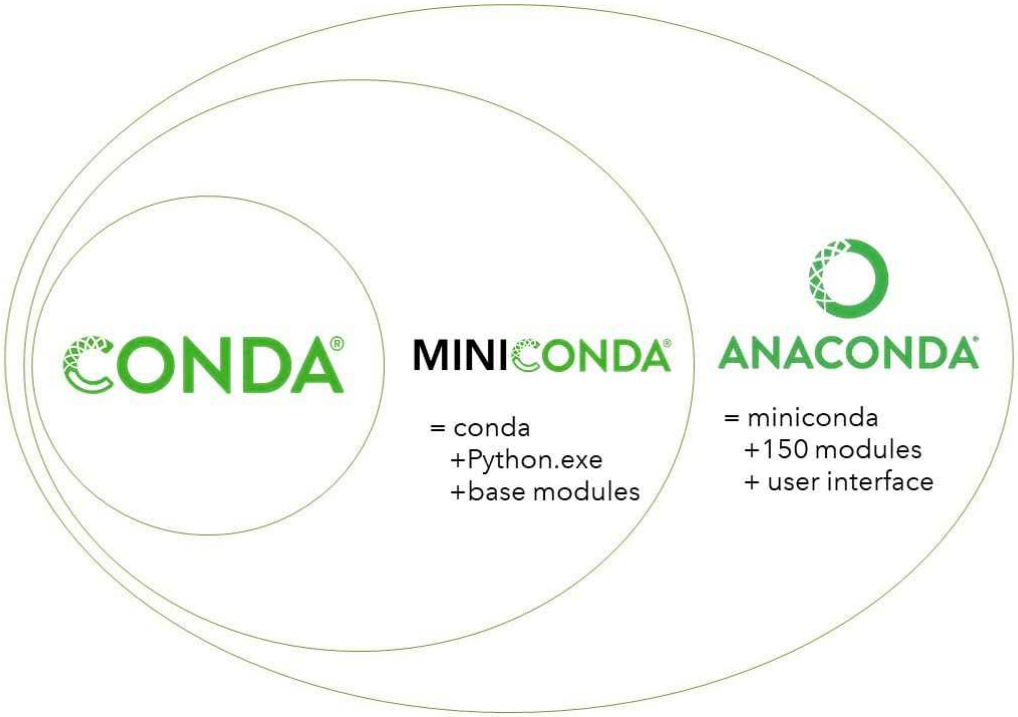
```
$ source ~/.bashrc    # set conda path
$ conda config --set auto_activate_base false
$ which conda
/scratch/$USER/miniconda3/condabin/conda
$ conda --version
conda 4.12.0
$ ls /scratch/$USER/miniconda3
./              conda-meta/  lib/         mkspecs/      qml/         ssl/
../             doc/         libexec/     phrasebooks/  resources/   translations/
bin/            envs/        LICENSE.txt  pkgs/         sbin/        var/
….
```

Anaconda

- **Anaconda**
  - distribution of the Python and R programming languages for scientific computing
    - data science, machine learning applications, large-scale data processing, predictive analytics, etc
  - aims to simplify package management and deployment
- **Conda**
  - open source package management system and environment management system
  - runs on Windows, macOS, Linux and z/OS
  - quickly installs, runs and updates packages and their dependencies.
- **PIP**
  - package installer for Python
  - use pip to install packages from the Python Package Index and other indexes.

Miniconda

- **a small, bootstrap version of Anaconda**
  - includes only conda, Python, the packages they depend on
- **a free minimal installer for conda**



Anaconda vs Miniconda

- **Number of packages**
  - Anaconda comes with over 150 data science packages, whereas miniconda comes with only a handful
- **Interface**
  - Anaconda has a graphical user interface (GUI) called the Navigator, while miniconda has a command-line interface

```
### Anaconda Download
$ wget https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86_64.sh

### Miniconda Download
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Conda command

clean	Remove unused packages and caches.
config	Modify configuration values in .condarc. This is modeled after the git config Writes to the user .condarc file (/home01/userID/.condarc) by default.
create	Create a new conda environment from a list of specified packages.
help	Displays a list of available conda commands and their help strings.
info	Display information about current conda install.
init	Initialize conda for shell interaction. [Experimental]
install	Installs a list of packages into a specified conda environment.
list	List linked packages in a conda environment.
package	Low-level conda package utility. (EXPERIMENTAL)
remove	Remove a list of packages from a specified conda environment.
uninstall	Alias for conda remove.
run	Run an executable in a conda environment. [Experimental]
search	Search for packages and display associated information. The input is a MatchSpec, a query language for conda packages. See examples below.
update	Updates conda packages to the latest compatible version.
upgrade	Alias for conda update

Horovod Framework

Horovod

- **distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet**
  - developed by Uber in 2017
- **aims to make distributed deep learning fast and easy to use**

Why Horovod

- neutral to DL frameworks to be used
  - ✓ Is it possible to make your DL codes run in parallel, irrespective of whether you use Tensorflow, Keras or Pytorch frameworks?
- easy to use & codify
  - ✓ How much modification does one have to make to a existing DL code to make it distributed?
  - ✓ How easy is it to run it?
- fast to run
  - How much faster would it run in distributed mode?
  - how easy is it to scale up?

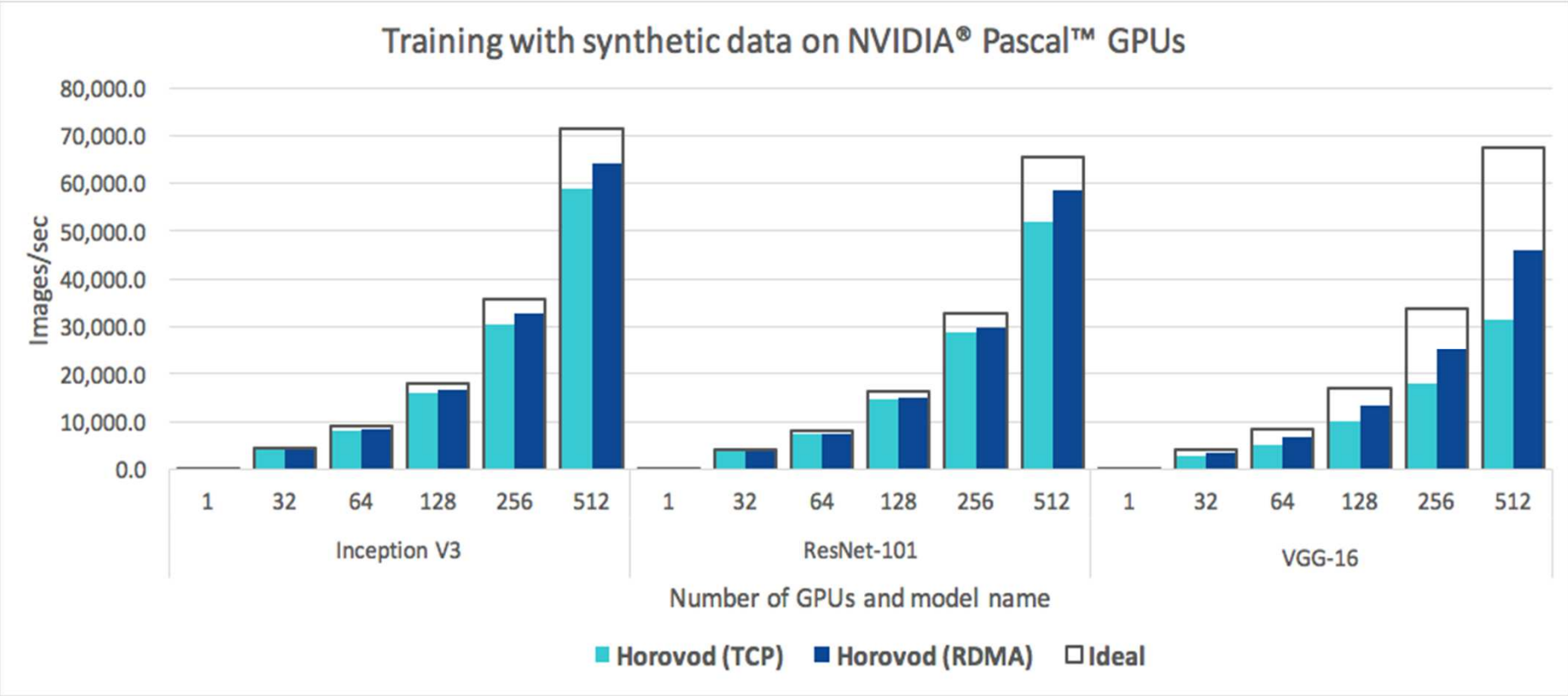
Why Horovod?

- **The primary motivation for this project is to make it easy to take a single-GPU training script and successfully scale it to train across many GPUs in parallel**
  - ✓ How much modification does one have to make to a program to make it distributed, and how easy is it to run it?
  - ✓ How much faster would it run in distributed mode?

- **Internally at Uber we found the MPI model to be much more straightforward and require far less code changes than previous solutions such as Distributed TensorFlow with parameter servers.**

Why Horovod?

- easy to use and fast
- scalling with Horovod



Horovod Usage

- **5 steps/lines to be added in your code**
  - Initialize Horovod
  - Pin GPU to each worker
  - Wrap the optimizer
  - Synchronize state across workers
  - Checkpoint on the first worker

Initialize Horovod

- **Tensorflow**
  - import horovod.tensorflow as hvd
  - hvd.init()
- **Kera**
  - import horovod.keras as hvd
  - hvd.init()
- **Pytorch**
  - import horovod.torch as hvd
  - hvd.init()

Pin a GPU for each worker

- **Tensorflow**
  - tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
- **Keras**
  - tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
- **Pytorch**
  - torch.cuda.set_device(hvd.local_rank())

Adjust learning rate and wrap the optimizer

- **Tensorflow**
  - opt = tf.optimizers.Adam(0.01 * hvd.size())
  - opt = hvd.DistributedOptimizer(opt,…)
- **Keras**
  - opt = keras.optimizers.Adadelta(0.01 * hvd.size())
  - opt = hvd.DistributedOptimizer(opt,…)
- **Pytorch**
  - opt = optim.SGD(model.parameters(), 0.01 * hvd.size())
  - opt= hvd.DistributedOptimizer(opt, …)

Synchronize states across workers

- **Tensorflow/Keras**
  - callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
- **Pytorch**
  - hvd.broadcast_parameters(model.state_dict(), root_rank=0)
  - hvd.broadcast_optimizer_state(optimizer, root_rank=0)

- **Ensure all workers start with the same weights**

Checkpoint on the first worker (rank 0)

- **Tensorflow/Keras**
  - if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint(args.checkpoint_format))
- **Pytorch**
  - if hvd.rank() == 0:

```
state = {'model': model.state_dict(),
         'optimizer': optimizer.state_dict(), }
torch.save(state, filepath)
```

Pytorch Example

```
import torch
import horovod.torch as hvd

# Initialize Horovod
hvd.init()

# Horovod: pin GPU to local rank.
torch.cuda.set_device(hvd.local_rank())

# Build model.
model = Net()
model.cuda()
optimizer = optim.SGD(model.parameters())

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer,
    named_parameters=model.named_parameters())
```

```
# Horovod: broadcast parameters.
hvd.broadcast_parameters(
    model.state_dict(),
    root_rank=0)

for epoch in range(100):
  for batch_idx, (data, target) in enumerate(...):
      optimizer.zero_grad()
      output = model(data)
      loss = F.nll_loss(output, target)
      loss.backward()
      optimizer.step()
```

Horovod execution command

- **MPI takes care of launching processes on all nodes**
- **Run on a node with 4-GPUs**
  - $ salloc …
  - $ mpirun -np 4 -H localhost:4 python train_hvd.py
  - $ horovodrun -np 4 -H localhost:4 python train_hvd.py
  - $ srun -n python train_hvd.py
- **Run on 4 nodes with 4-GPUs each node**
  - $ salloc …
  - $ mpirun/horovodrun -np 4 -H node1:4,node2:4,node3:4,node4:4 python train_hvd.py
  - $ srun -n 16 python train_hvd.py

Horovod Installation on Neuron

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 cmake/3.16.9
$ conda create -n horovod
$ conda activate horovod
$ conda install pytorch==1.12.0 torchvision==0.13.0 torchaudio==0.12.0 cudatoolkit=11.3 -c pytorch
$ pip install tensorflow-gpu==2.10.0
$ HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_WITH_TENSORFLOW=1
HOROVOD_WITH_PYTORCH=1 \
HOROVOD_WITH_MPI=1 HOROVOD_WITH_GLOO=1 pip install --no-cache-dir horovod
$ horovodrun -cb
Horovod v0.26.1:

Available Frameworks:
    [X] TensorFlow
    [X] PyTorch
    [ ] MXNet

Available Controllers:
    [X] MPI
    [X] Gloo

Available Tensor Operations:
    [X] NCCL
    [ ] DDL
    [ ] CCL
    [X] MPI
    [X] Gloo
```

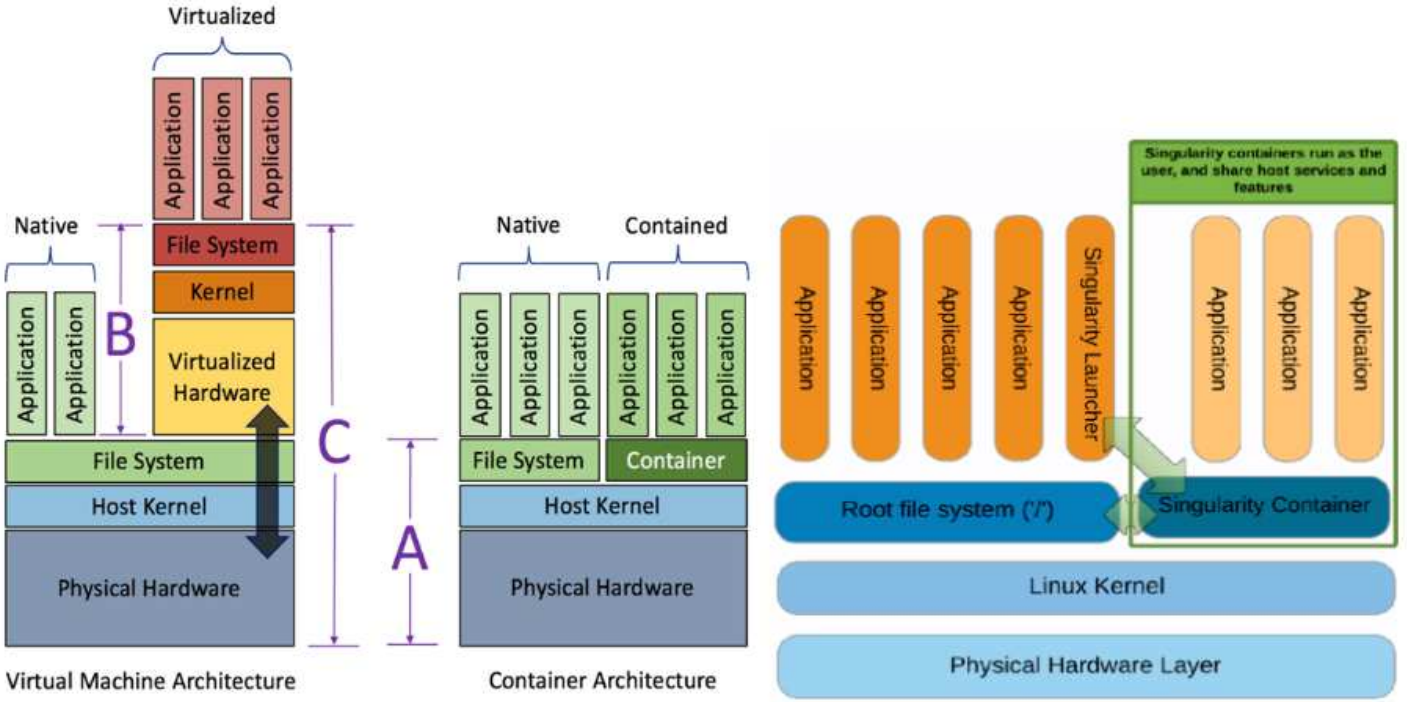
Horovod execution on Neuron

```
1) request an allocation of available on neuron
$ salloc --partition=amd_a100nv_8 -J debug --nodes=2 --time=2:00:00 --
gres=gpu:4 --comment=horovod

2) load modules
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 cmake/3.16.9

3) activate the horovod virtual environment
$ conda activate horovod

4) run horovod applications
(horovod) $ mpirun/horovod -np 8 -H gpu#:4,gpu#:4 python train_hvd.py
(horovod) $ srun -n python train_hvd.py
```

Singularity Container

Singularity

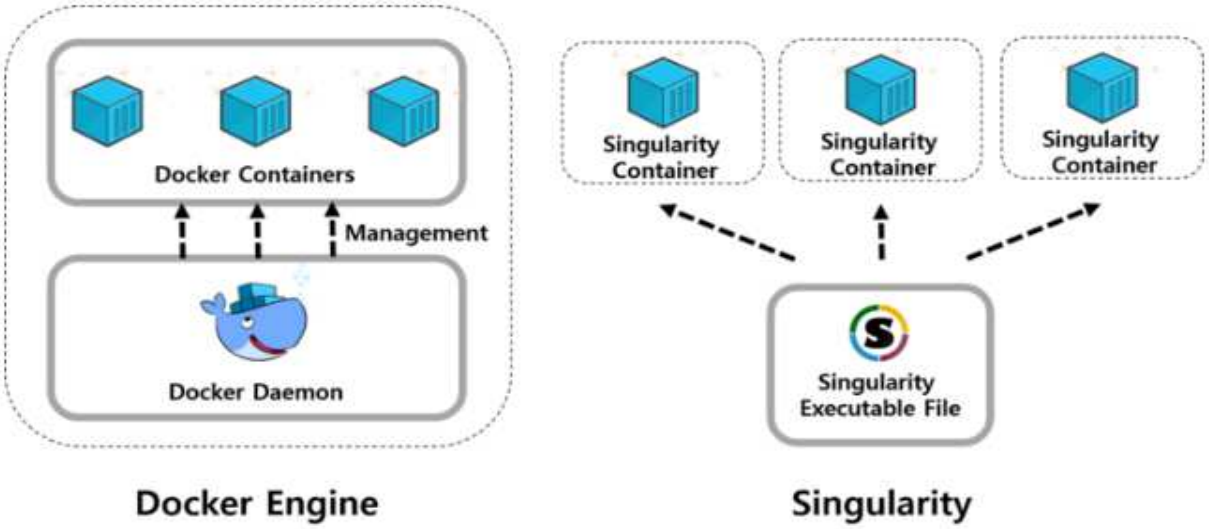
- **a container platform for HPC**



Container-based applications have *direct access* to the host kernel and hardware and, thus, are able to achieve similar performance to native applications. In contrast, VM-based applications only have *indirect access* via the guest OS and hypervisor, which creates a significant performance overhead.

Why Singularity?

- **A container platform for HPC**
  - Each container is a single image file
  - No root owned daemon processes
  - Support shared/multi-tenant resource environment
  - Support HPC hardware
    - Infiniband, GPUs
  - Support HPC applications
    - MPI



Running Horovod using Singularity on Neuron

- **No bother to deal with conda & horovod**
- **Just allocate nodes using salloc and run singularity container. That's it!!!**

```
$ salloc --partition=amd_a100nv_8 -J debug --nodes=2 --time=2:00:00 --
gres=gpu:4 --comment=pytorch

$ srun -n 8 singularity exec --nv
/apps/applications/singularity_images/ngc/pytorch_22.03-hd-py3.sif python
pytorch_imagenet_resnet50.py
/usr/bin/rm: cannot remove '/usr/local/cuda/compat/lib': Read-only file system
/usr/bin/rm: cannot remove '/usr/local/cuda/compat/lib': Read-only file system
…….
Train Epoch     #10:   0%|          | 1/5005 [00:13<19:03:52, 13.72s/it, loss=2.29, accurTrain Epoch
#10:   0%|          | 1/5005 [00:13<19:03:52, 13.72s/it, loss=2.22, accurTrain Epoch     #10:   0%|          |
2/5005 [00:13<19:03:38, 13.72s/it, loss=2.16, accurTrain Epoch     #10:   0%|          | 3/5005
[00:13<5:00:52,  3.61s/it, loss=2.16, accuraTrain Epoch     #10:   0%|          | 3/5005 [00:13<5:00:52,
3.61s/it, loss=2.17, accuraTrain Epoch     #10:   0%|          | 4/5005 [00:13<5:00:48,  3.61s/it, loss=2.23,
accura
……..
```

Singularity Usage on Neuron

- **Web site: https://www.ksc.re.kr/gsjw/jcs/hd**

- **job script directory**
  - /apps/applications/singularity_images/examples
- **Singularity Container Images directory**
  - /apps/applications/singularity_images/ngc
- **Pytorch examples directory**
  - Single node
    - /apps/applications/singularity_images/examples/pytorch/resnet50v1.5
  - Multiple nodes
    - /apps/applications/singularity_images/examples/horovod/examples/pytorch
- **Imagenet datasets directory**
  - Training data
    - /apps/applications/singularity_images/imagenet/train
  - Validation data
    - /apps/applications/singularity_images/imagenet/val

Github Site

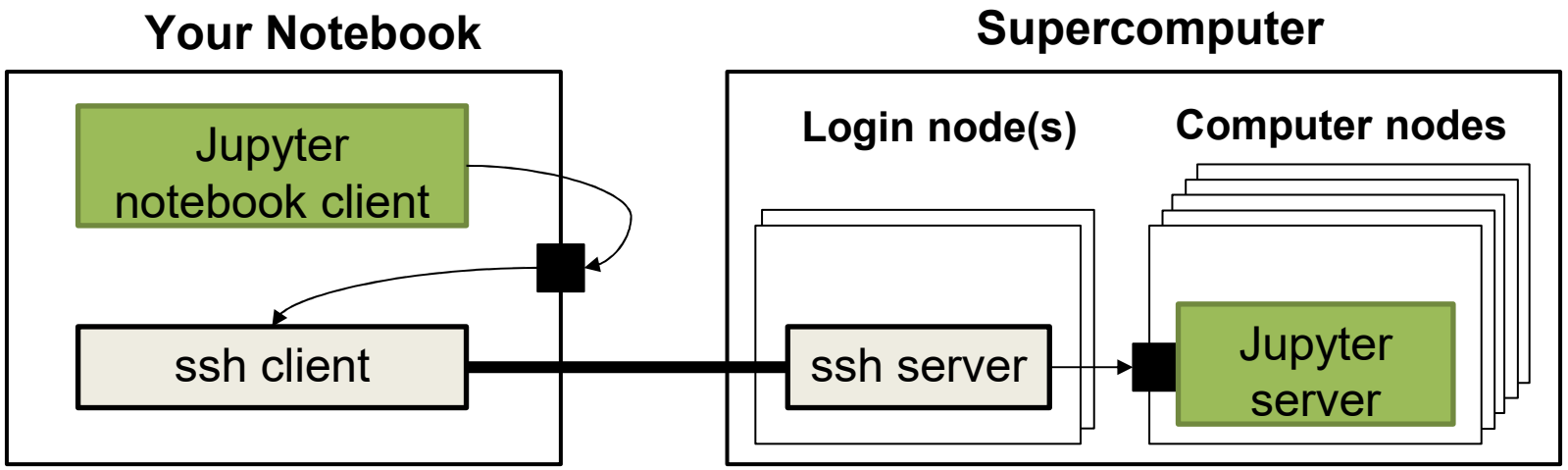
- **KISTI Neuron cluster**
  - https://github.com/hwang2006/KISTI-DL-tutorial-using-horovod
- **NERSC Perlmutter Supercomputer**
  - https://github.com/hwang2006/distributed-training-on-perlmutter-using-horovod
- **Distributed Training on Supercomputer with Pytorch Lightning**
  - https://github.com/hwang2006/distributed-training-on-supercomputer-with-pytorch-lightning
- **NLP Practices on Supercomputer with Pytorch Lightnign**
  - https://github.com/hwang2006/NLP-practices-on-supercomputer-with-pytorch-lightning

Thank you

Contact: Soonwook Hwang <hwang@kisti.re.kr>

SSH Tunneling (SSH Port Forwarding)

- **Local Tunneling (Local Port Forward)**
  - ssh -L localhost:8888:gpu31:21441 qualis@neuron.ksc.re.kr



R&D혁신지원 프로그램

- **R&D 혁신지원프로그램 안내 및 신청**
  - https://www.ksc.re.kr/jwjg/hsjw/hsjwan (안내)
  - https://www.ksc.re.kr/jwjg/hsjw/hsjwsc/list (신청)

R&D 혁신지원 프로그램 신청

KISTI는 초고성능컴퓨팅(High Performance Computing, 이하 HPC) 육성법에 근거하여 국가 차원의 초고성능컴퓨팅을 육성하고 있습니다.
이에 KISTI 국가슈퍼컴퓨팅본부는 국내 계산과학공학분야 연구자에게 혁신지원 프로그램을 통해 초고성능컴퓨팅 자원을 무상으로 제공해 왔습니다.
국가 초고성능컴퓨팅 육성 기본계획에 명시된 자원 배분 정책에 따르고 있으며, 2023년도는 거대연구 및 창의연구 두 분야로 나누어 지원 중에 있습니다.

2023년 연간일정

차수	신청서 접수 기간	평가 실시 기간	지원 기간	보고서 제출 기간	논문 투고 기간	논문 실적 제출 기간
2023년 2차	3.2 ~ 3.16	3.22 ~ 4.19	23.5.1 ~ (1년간)	지원종료 후 1개월 이내	지원종료 후 6개월 이내	지원종료 후 2년 이내
2023년 3차	7.1 ~ 7.14	7.19 ~ 7.27	23.9.1 ~ (1년간)			
2024년 1차	11.1 ~ 11.14	11.18 ~ 11.26	24.1.1 ~ (1년간)			

Timeline of Large Language Models (LLMs)

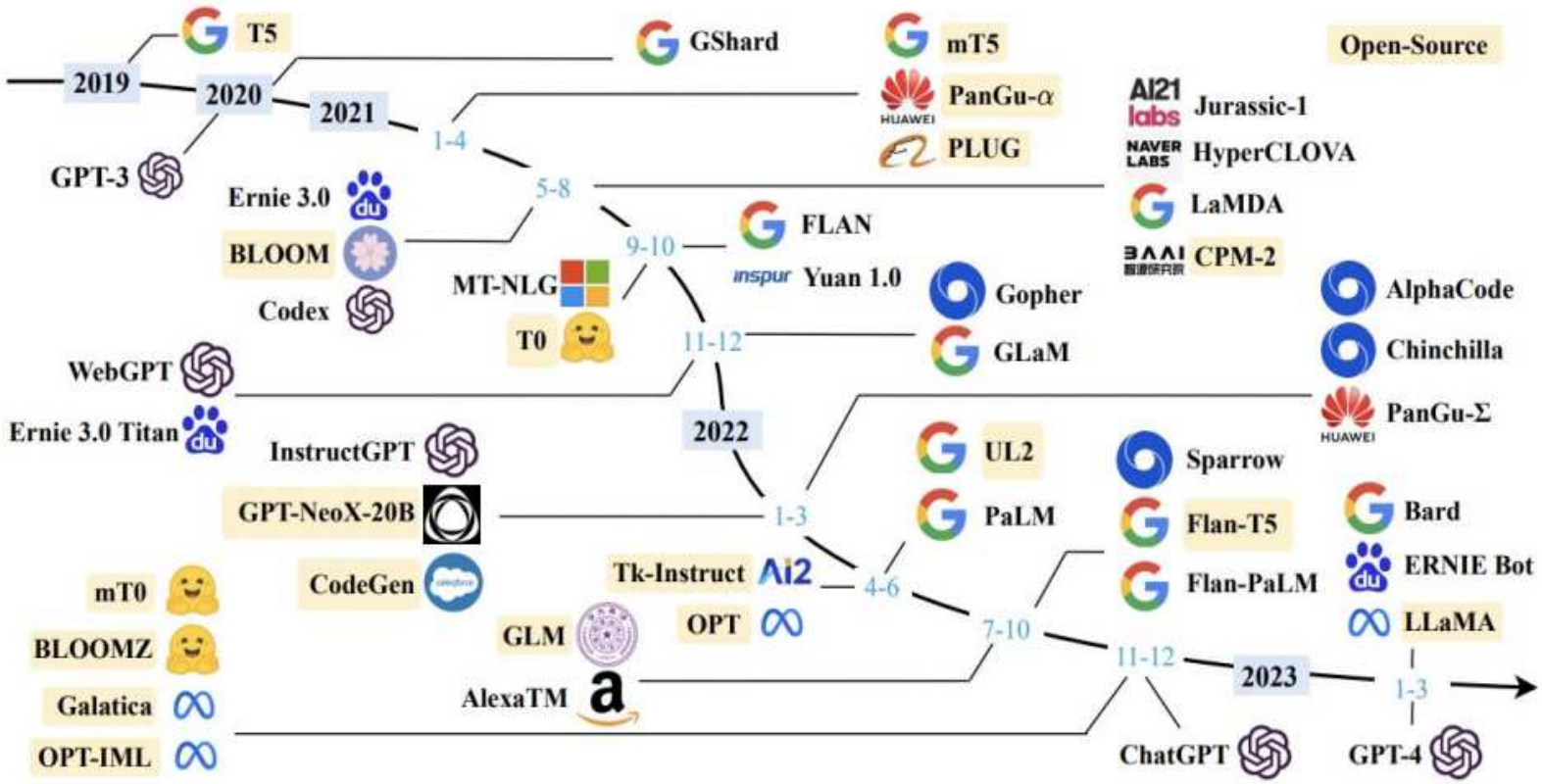


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

A Survey of Large Language Model
https://arxiv.org/pdf/2303.18223.pdf

T5: Scaling

- **medel size matters most**

Scaling strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× size, 4× training steps	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× size, 4× batch size	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× size, 2× training steps	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× size, 1× training steps	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled, fine-tune only	84.05	19.57	82.36	71.55	27.55	40.22	28.09

Exploring the Limits of Transfer Learning with a Unified Text- to- Text Transformer
https://arxiv.org/abs/1910.10683