# Distributed Deep Learning with Tensorflow2, PyTorch2 and Horovod

Pierrick Pochelu – PCOG Team / HLST

# Outline

- What is Horovod ?
- How to adapt my Tensorflow/PyTorch code for using Horovod ?
- Practical session

# What is Horovod ?

uni.lu

# Introduction to Horovod

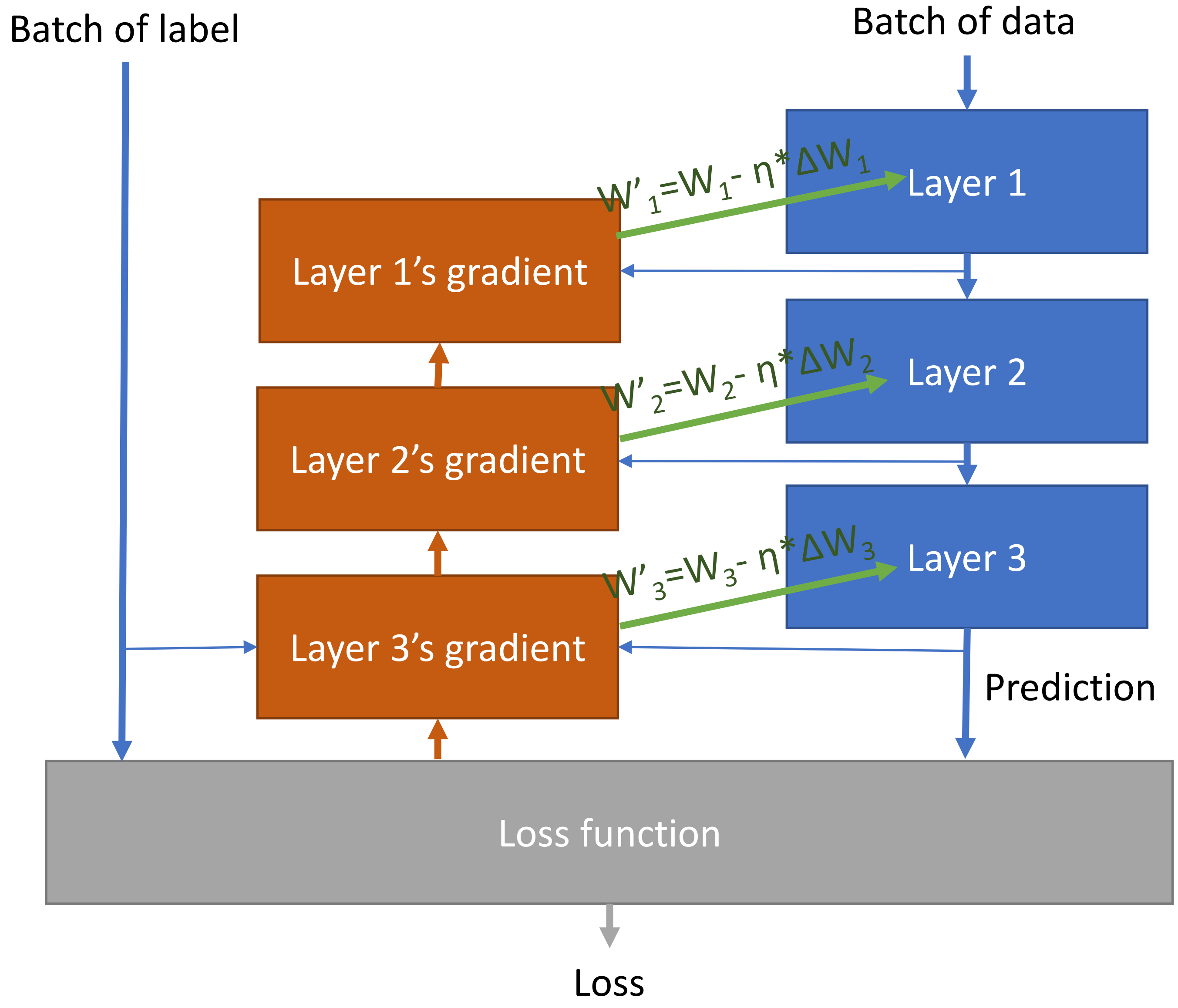
- Distributed Deep Learnining training across multi-machine multi-GPU

- Open-source:
  - Linux AI Foundation

- Framework agnostic
  - Tensorflow2, Keras2, PyTorch2, MXNet

- Last GPU HPC techniques:
  - NCCL: inter-GPU communication (in one machine) by-passing the CPU
  - GPUDirect RDMA: inter-machine communication by-passing the CPU and intermediate storage
  - Tensor Fusion: Aggregate layer’s tensor in 1 tensor before AllReduce

# Stochastic Gradient Descent computing graph

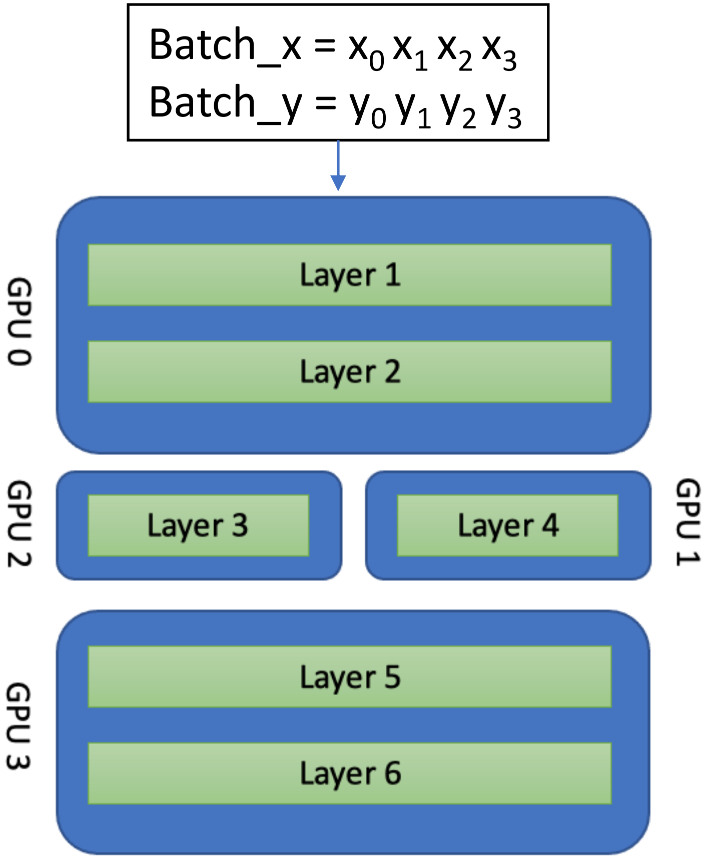


For each batch of data:

1) Forward
2) Backward
3) Update

# Parallel SGD

## Model Parallelism



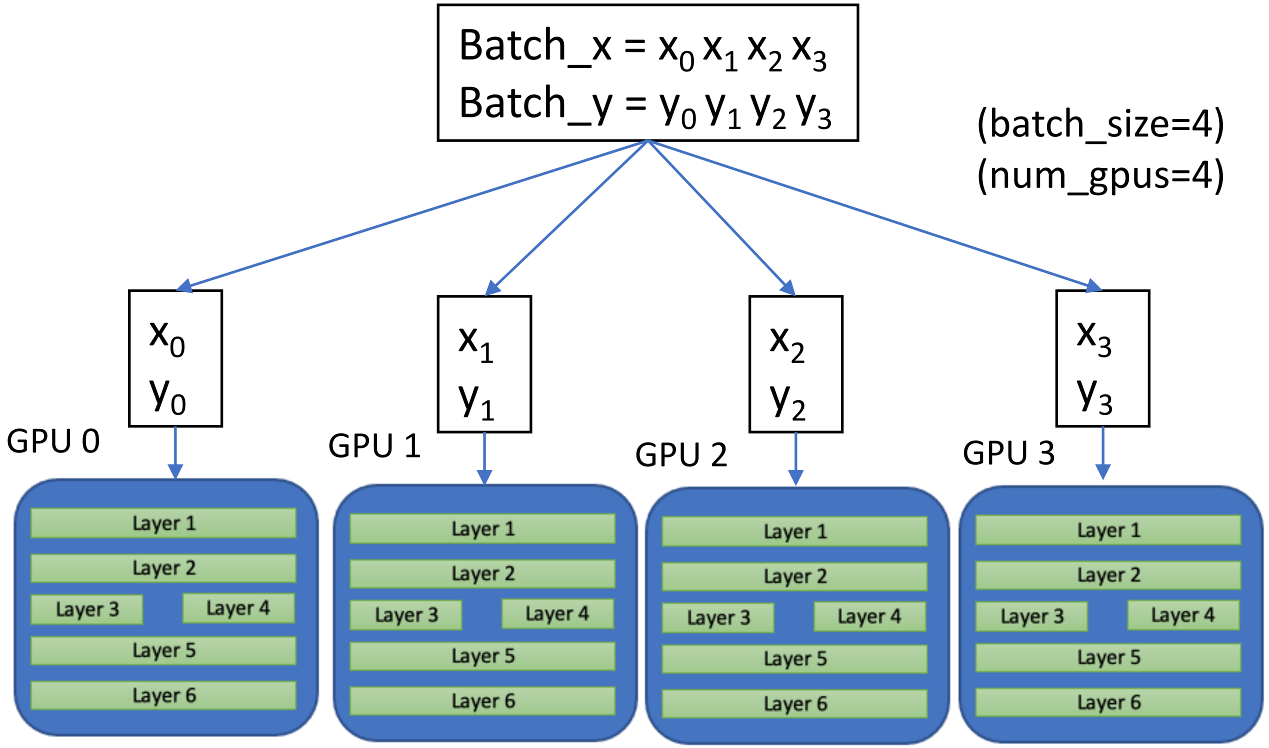
Batch_x = $x_0\, x_1\, x_2\, x_3$

Batch_y = $y_0\, y_1\, y_2\, y_3$

**Useful when:**

**Model** too large for a single GPU

## Data Parallelism



Batch_x = $x_0\, x_1\, x_2\, x_3$

Batch_y = $y_0\, y_1\, y_2\, y_3$

(batch_size=4)

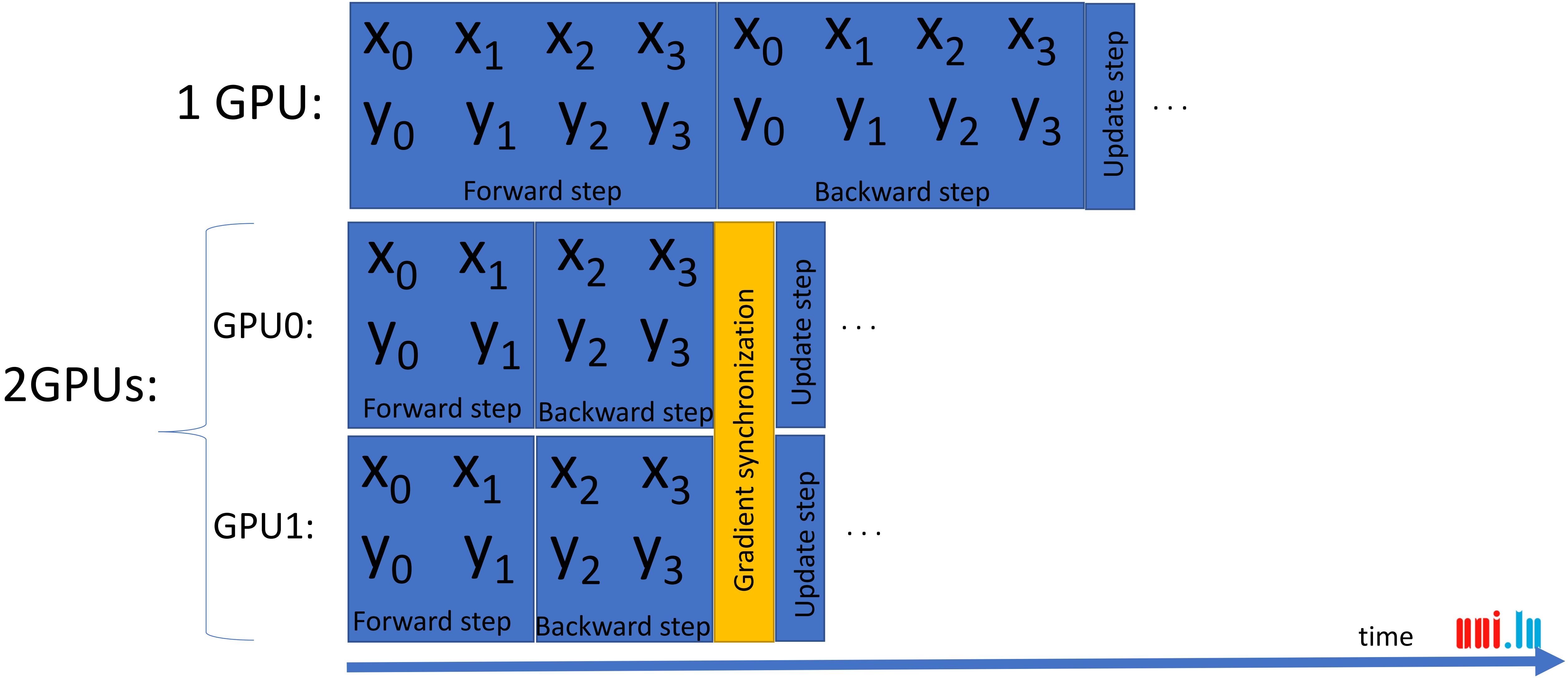
(num_gpus=4)

**Useful when:**

**Batch of data** too large for a single GPU

**Speed up** by splitting batch workload across GPUs

# Data parallel SGD function of time



1 GPU:

$x_0$ $x_1$ $x_2$ $x_3$ $y_0$ $y_1$ $y_2$ $y_3$ Forward step

$x_0$ $x_1$ $x_2$ $x_3$ $y_0$ $y_1$ $y_2$ $y_3$ Backward step

Update step

. . .

2GPUs:

GPU0:

$x_0$ $x_1$ $y_0$ $y_1$ Forward step

$x_2$ $x_3$ $y_2$ $y_3$ Backward step

Update step

. . .

GPU1:

$x_0$ $x_1$ $y_0$ $y_1$ Forward step

$x_2$ $x_3$ $y_2$ $y_3$ Backward step

Update step

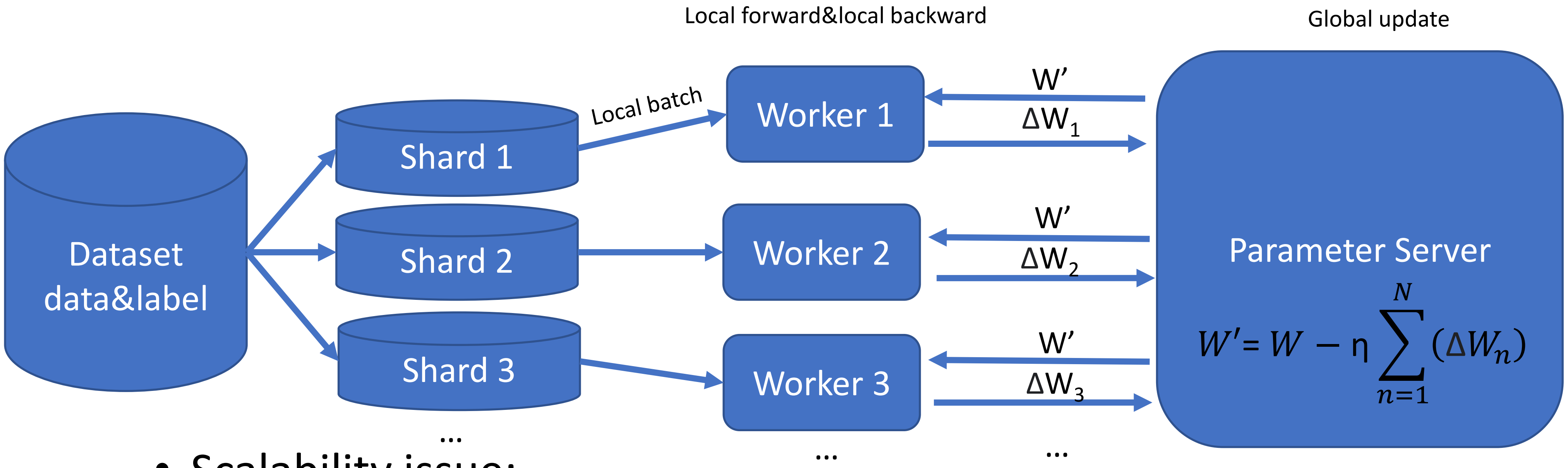
. . .

Gradient synchronization

time

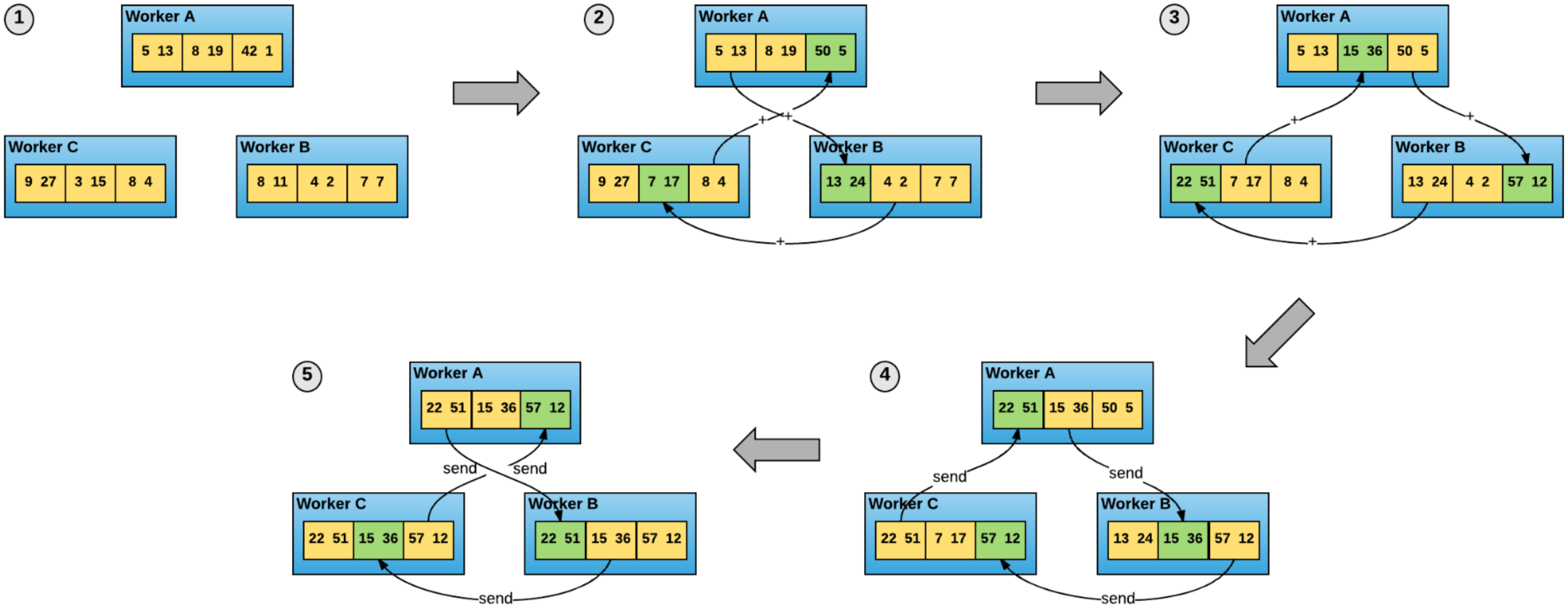
# Design1:
# Data Parallel SGD with Parameter Server

- Using a parameter server:



- Scalability issue:
  - Parameter server computation may be the bottlneck
  - Parameter server communication: many-to-one and one-to-many
- *Asynchronous* SGD with stale gradients → improve the computing speed → but hurts convergence

# Design 2:
# Data Parallel SGD using Ring All-Reduce



Better scalability:

- Fair computing between workers
- Fair usage of communication links

“Horovod: fast and easy distributed deep learning in TensorFlow”
https://arxiv.org/pdf/1802.05799.pdf

# How to adapt my code for using Horovod ?

# Updating your code for using Horovod

1. Initialization
2. Compute « local_batch_size »
3. Pinning the process to the GPU
4. Sharding data
5. Initialize model weights to all workers
6. Gradient communication callback

# Horovod code with PyTorch2

1. Initialization

```
import horovod.torch as hvd
hvd.init()
```

2. Compute « local_batch_size »

```
local_batch_size = BATCH_SIZE // int(hvd.size())
```

3. Pinning the process to the GPU

```
if torch.cuda.is_available():
    torch.cuda.set_device(hvd.local_rank()) # Horovod: pin GPU to local rank.
    torch.cuda.manual_seed(42)
    kwargs = {"num_workers": 1, "pin_memory": True}
else:
    kwargs = {}
torch.set_num_threads(1) num. of CPU threads to be used per worker
```

4. Sharding data

```
torch_sampler=torch.utils.data.distributed.DistributedSampler(torch_dataset,
                                                              num_replicas=hvd.size(),
                                                              rank=hvd.rank() )
torch_loader = torch.utils.data.DataLoader(torch_dataset,
                                           batch_size=local_batch_size,
                                           sampler=torch_sampler, **kwargs )
```

5. Initialize model weights to all workers

```
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
```

6. Gradient communication callback

```
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     op=hvd.Average,
                                     gradient_predivide_factor=1)
```

# Horovod code with Tensorflow2

1. Initialization

```
import horovod.tensorflow.keras as hvd
hvd.init()
```

2. Compute « local_batch_size »

```
local_batch_size = BATCH_SIZE // int(hvd.size())
```

3. Pinning the process to the GPU

```
gpus = tf.config.experimental.list_physical_devices("GPU")
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], "GPU")
```

4. Sharding data

```
if int(hvd.size()) > 1:
    num_train_per_replica = len(X_train) // int(hvd.size())
    X_train = X_train[
        int(hvd.rank()) * num_train_per_replica :
        (int(hvd.rank()) + 1) * num_train_per_replica ]
    Y_train = Y_train[
        int(hvd.rank()) * num_train_per_replica :
        (int(hvd.rank()) + 1) * num_train_per_replica ]
```

5. Initialize model weights to all workers

```
callbacks = [ hvd.callbacks.BroadcastGlobalVariablesCallback(0) ]
```

6. Gradient communication callback

```
optimizer = tf.optimizers.Adam(LEARNING_RATE)
optimizer = hvd.DistributedOptimizer(
    optimizer, backward_passes_per_step=1, average_aggregated_gradients=True)
```

# Official code analysis

Keras:

https://github.com/horovod/horovod/blob/master/examples/keras/keras_mnist.py

- How the number of GPUs affects the result ?
- How many time the entire dataset is loaded in memory ?

PyTorch:

https://github.com/horovod/horovod/blob/master/examples/pytorch/pytorch_lightning_mnist.py

- How the number of GPUs affects the result ?
- How many time the entire dataset is loaded in memory ?
- Discuss some differences between the Keras code and the PyTorch code.

# Proposed TF2/PyTorch2 code

https://ulhpc-tutorials.readthedocs.io/en/latest/deep_learning/horovod/#horovod

# Practical session

# Practical session

```
si-gpu -G2 -t120 -c6 --reservation=hpcschool-gpu
cd /work/projects/ulhpc-tutorials/PS10-Horovod/
source env.sh

# Environment testing
pip list
horovodrun --check-build

# Launching a first Horovod test
mpirun -n 1 python test_horovod.py
mpirun -n 2 python test_horovod.py

mpirun -n 1 python tensorflow_horovod_basic.py # Notice the computing time
mpirun -n 2 python tensorflow_horovod_basic.py # Compare the time per epoch
# /!\ The first epoch is slower than the other one (still initializing)
```

# Practical session (may take >10 minutes)

```
mpirun -n 1 python tensorflow_horovod.py
mpirun -n 2 python tensorflow_horovod.py
# /!\ The first epoch is slower than the other one
(still initializing)


mpirun -n 1 python pytorch_horovod.py
mpirun -n 2 python pytorch_horovod.py
```

# see the output on: https://ulhpc-tutorials.readthedocs.io/en/latest/deep_learning/horovod/#horovod

# Multi-node multi-GPU

```
#!/bin/sh –l
#SBATCH -c 6
#SBATCH -N 2
#SBATCH -p gpu
#SBATCH --gpus-per-node 4
#SBATCH -t 120
#SBATCH --export=ALL

mpirun –n 8 python test_horovod.py
```

# Contact me ☺

If you want to accelerate your HPC/AI application.

Or any issue with Horovod.

Contact me: pierrick.pochelu@uni.lu

# Thank you for your attention

Any question ?