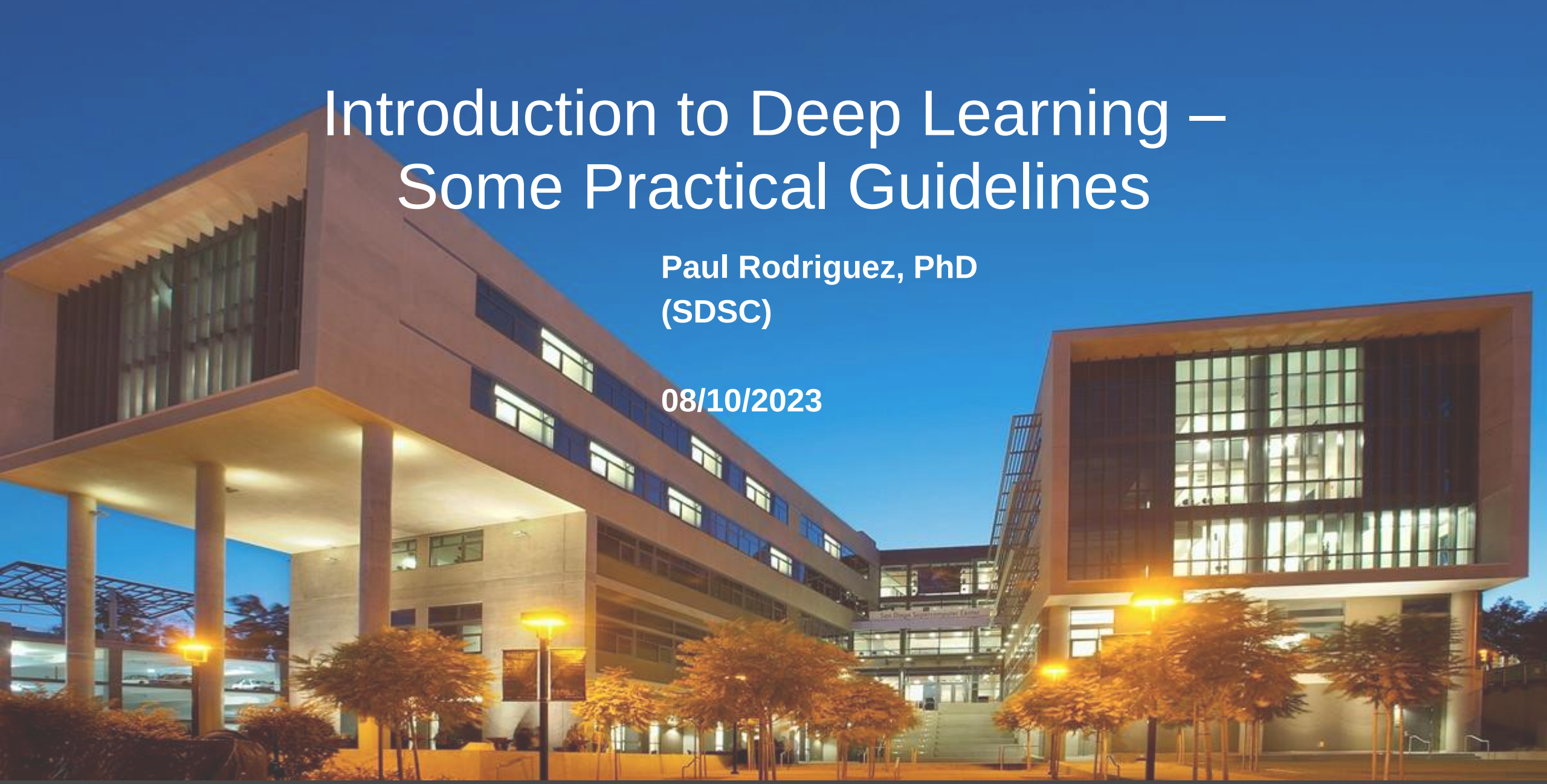
Introduction to Deep Learning – Some Practical Guidelines

Paul Rodriguez, PhD
(SDSC)

08/10/2023



Outline

- **Part IB - Practical Guidelines for Running a Project:**

Choosing Hyperparameters – a bit of exploration and exploitation

Job workflow - make it efficient and easy to organize

CPUs vs GPUs

Parallelize Models and Multinode Execution, with an exercise

Choosing Hyperparameters

- Hyperparameters are found by searching, not by the network algorithm

- Generally, hyperparameters related to:
  - architecture (layers, units, activation, filters, …)
  - algorithm (learning rate, optimizer, epochs, …)
  - efficient learning (batch size, normalization, initialization, …)

- Some options are determined by task:
  - loss function, CNN vs MLP, …

- Use what works, from related work or the latest recommendations,

Hyperparameters Search

- Can take a long time, hard to find global optimal
- Start with small data, short runs to get sense of range of good parameter values
- Easy but possibly time-consuming method:
  grid search over uniformly spaced values
- Do “exploration” then “exploitation”, ie search wide then search deep
  Keras Tuner functions can help with the wide search

Keras Hyperparameter Search Tool

- Keras Hypertuner class implements several search strategies:

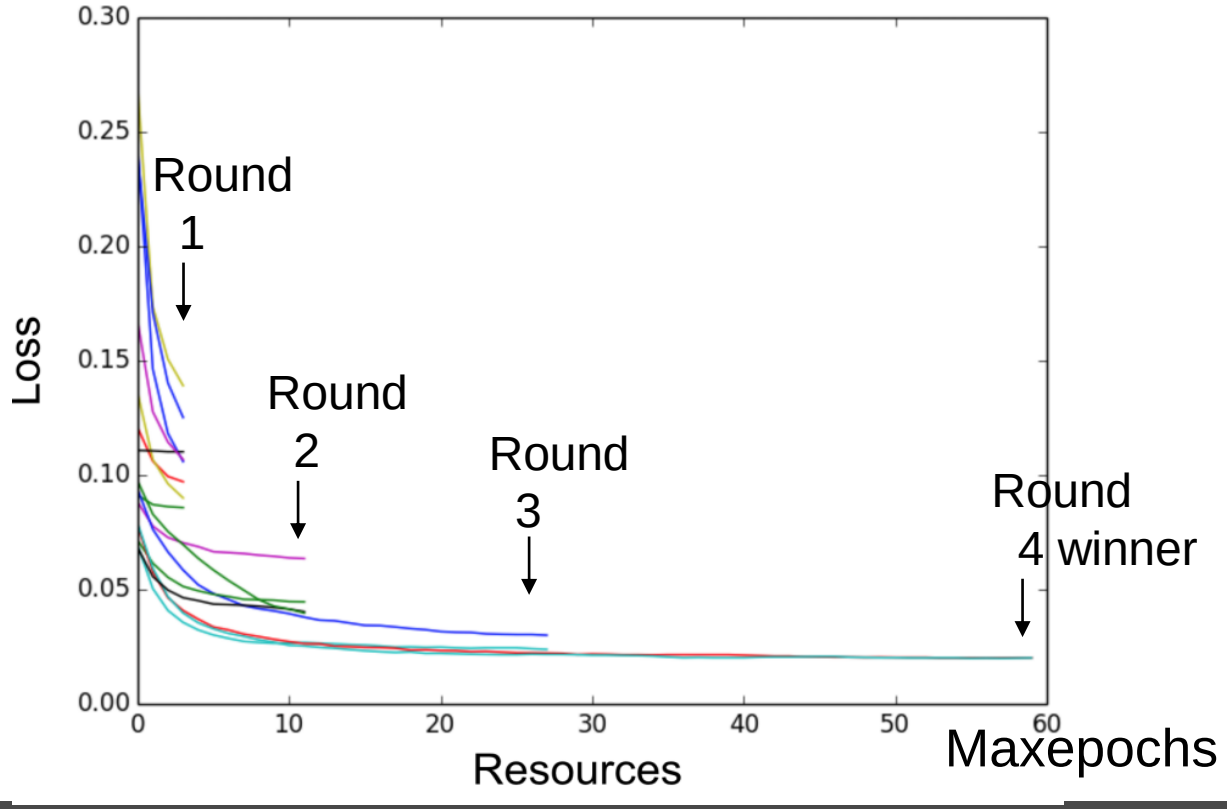
Hyperband is like a tournament of hyperparameter configurations, with incremental training, to weed out worse ones

RandomSearch will search randomly through the space of configurations and try to find better regions

Bayesian optimization is like function approximation to pick out next configuration

Hyperband Bracket

Each round runs several network configurations for small number of epochs
Several rounds with increasing epochs make up a bracket
Several brackets are run to end up with several possible overall winners.



Note, you could run a small grid search around hyperband winners to confirm performance

Workflow and Organizing Jobs

Job Level: What makes sense to include in each job?

Model Level: run & test model for each parameter configuration

Data Level: loop through cross validation datasets (if applicable)

- **Consider how long each a model runs for 1 configuration of hyperparameters for 1 dataset**
- **Organize jobs into reasonable chunks of work**
- **For large models consider model-checkpoints**
- **Tensorboard is available but needs to be secure (ask for details)**

Organizing Configurations – one way

Code snippet: using 'YAML' file to set up hyperparameter configuration

Create text file with "Parameter: Value" pairs

/home/p4rodrig/WKSHOPS...

```
filter_size: 16
num_filters: 16
```

Line: 1/2 Encoding: 1252 (ANSI - La

Read file as python dictionary

```python
import yaml

with open("./modelrun_args.yaml", "r") as f:
    my_yaml=yaml.safe_load(f)  #this returns a python dictionary

filter_size=my_yaml.get("filter_size")
num_filters=my_yaml.get("num_filters")
print('arguments, filter size:',filter_size,' num filters',num_filters)
```

Example slurm job script and execution for Expanse

```
#!/usr/bin/env bash
#SBATCH --job-name =mnist0522
#SBATCH --account=sds164
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --time=00:10:00
#SBATCH --output=myjoboutput.o%j.%N.out

module purge
module load singularitypro
module list

echo "filter_size: 3 " >  modelrun_args.yaml
echo "num_filters: 16 " >> modelrun_args.yaml

singularity exec --bind /expanse,/scratch  --nv \
  /cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.s
  python3 Intro_mnist_cnn2_forbatch.py > mymnist_stdoutput.txt
```

You could also modify or set up parameters; and save yaml files for each run

note on using GPU

- **GPU node has multiple GPU devices**
- **By default tensforflow will run on 0th gpu device if GPU is available, otherwise it will use all CPU cores**

note on using GPU

- **GPU node has multiple GPU devices**
- **By default tensforflow will run on 0th gpu device if GPU is available, otherwise it will use all CPU cores**

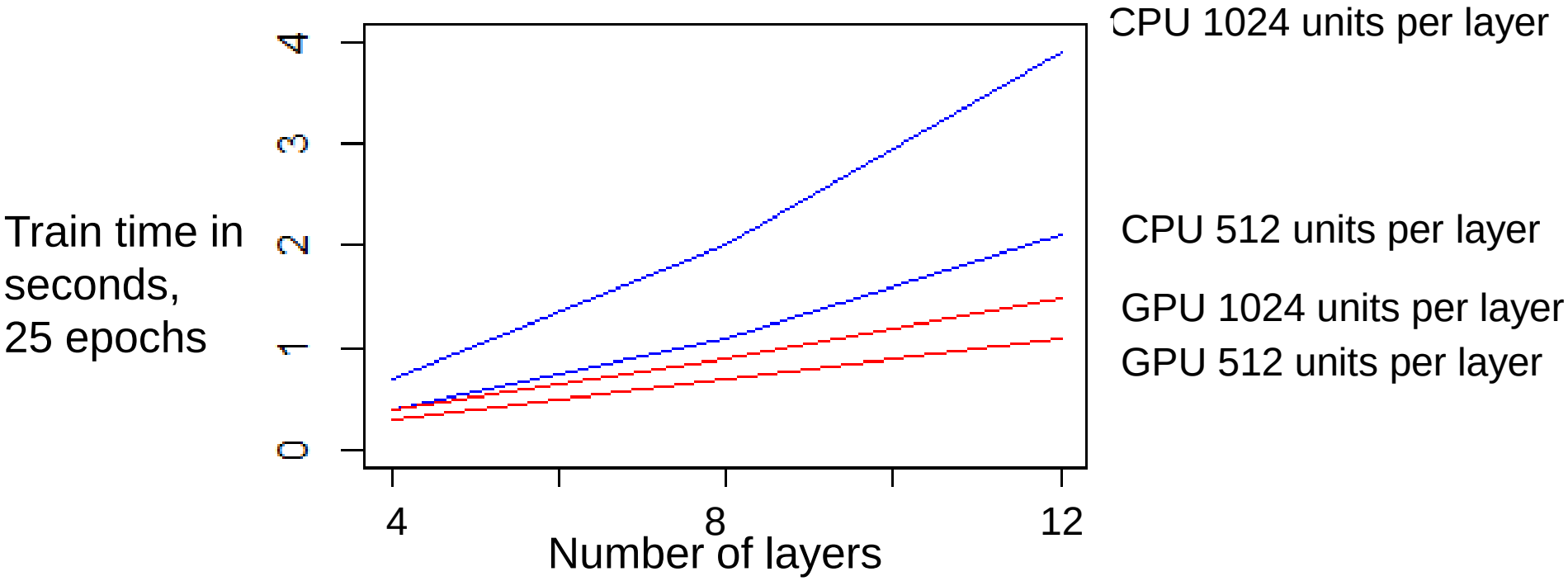
Code snippet to check for GPU devices

/home/p4rodrig/WKSHOPS/EXP-05192022/Intro_mnist_cnn2_forbatch.py - p4rodrig@login.expanse.sdsc.edu - Editor - W

Encoding Color

```
physical_devices = tf.config.list_physical_devices('GPU')
n_gpus           = len(physical_devices)
print("Info,config,Num GPUs:", n_gpus)
if tf.test.gpu_device_name():
  print('Info, test,Default GPU Device:{}'.format(tf.test.gpu_device_name()))
```

GPU shared (V100) vs CPU (128 cores)
For MLP with Dense Layers, 80000x200 data matrix



GPUs faster, but you might have to wait more in job queue; also some memory limits compared to CPU, may need to use smaller batch size

Parallel DL models with multiple nodes/devices

- **The main approach to parallelize training: Data Parallel:**
    1. Split up data (in Keras see ‘tf.data.Datasets’ API or use numpy)
    2. Launch your script on each device
    3. Each device trains a copy of the model with a part of the data
    4. Aggregate parameter updates across model instances

Parallel DL models with multiple nodes/devices

- **The main approach to parallelize training: Data Parallel:**
    1. Split up data
    2. Launch your script on each device
    3. Each device trains a copy of the model with a part of the data
    4. Aggregate parameter updates

- **Main tools: Keras/Tensorflow 'strategy' or use Horovod MPI wrappers**

- **Other approaches include Model Parallel (e.g. few layers per device), or Model and Data parallel**

- **Also, using mixed precision can reduce memory footprint**

Keras/Tensorflow strategy single GPU node

Set up a 'mirror' strategy

```
mirrored_strategy = tf.distribute.MirroredStrategy(["GPU:0", "GPU:1",  "GPU:2",  "GPU:3"])
```

You also need the strategy scope around the model definition so that it can make copies

```
if (n_gpus>0):
    with mirrored_strategy.scope():
      multi_dev_model=build_model()
```

Then train as normal (use batch size multiple of 32)

Keras/Tensorflow strategy multiple GPU node

Keras also has a ‘multiworker’ strategy but it requires setting up config files with IP addresses

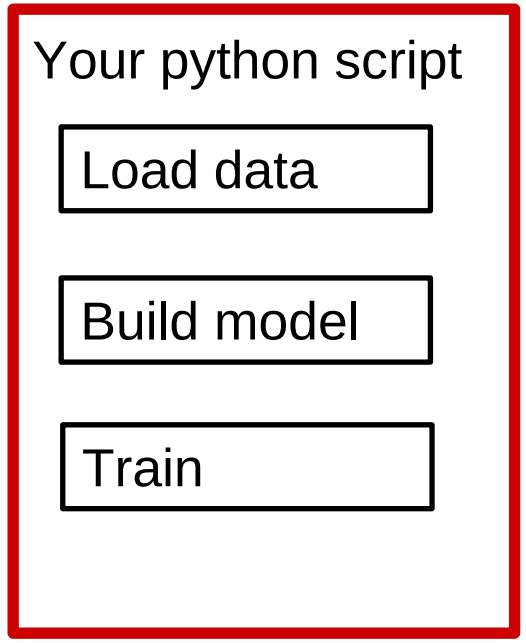
But, on HPC systems resources are shared so IP addresses are dynamic

Thus, it is better to use Horovod with MPI and slurm batch job

For example, single node, single device execution

In slurm batch script:

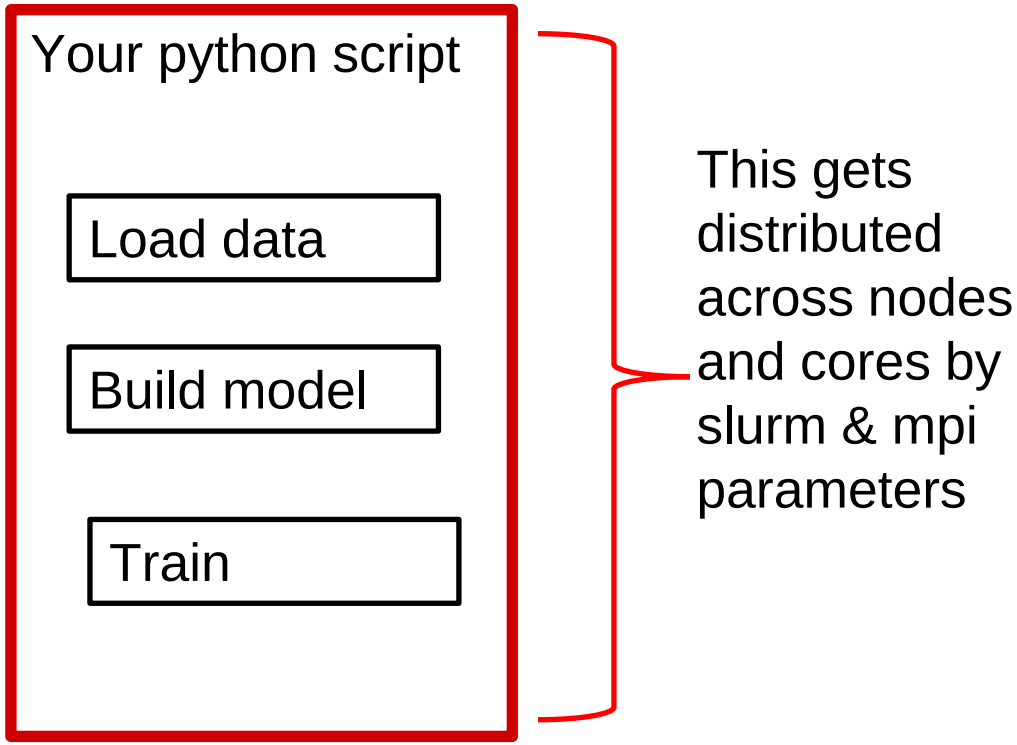
singularity → python



Multinode, mpi launches instances

In slurm batch script:

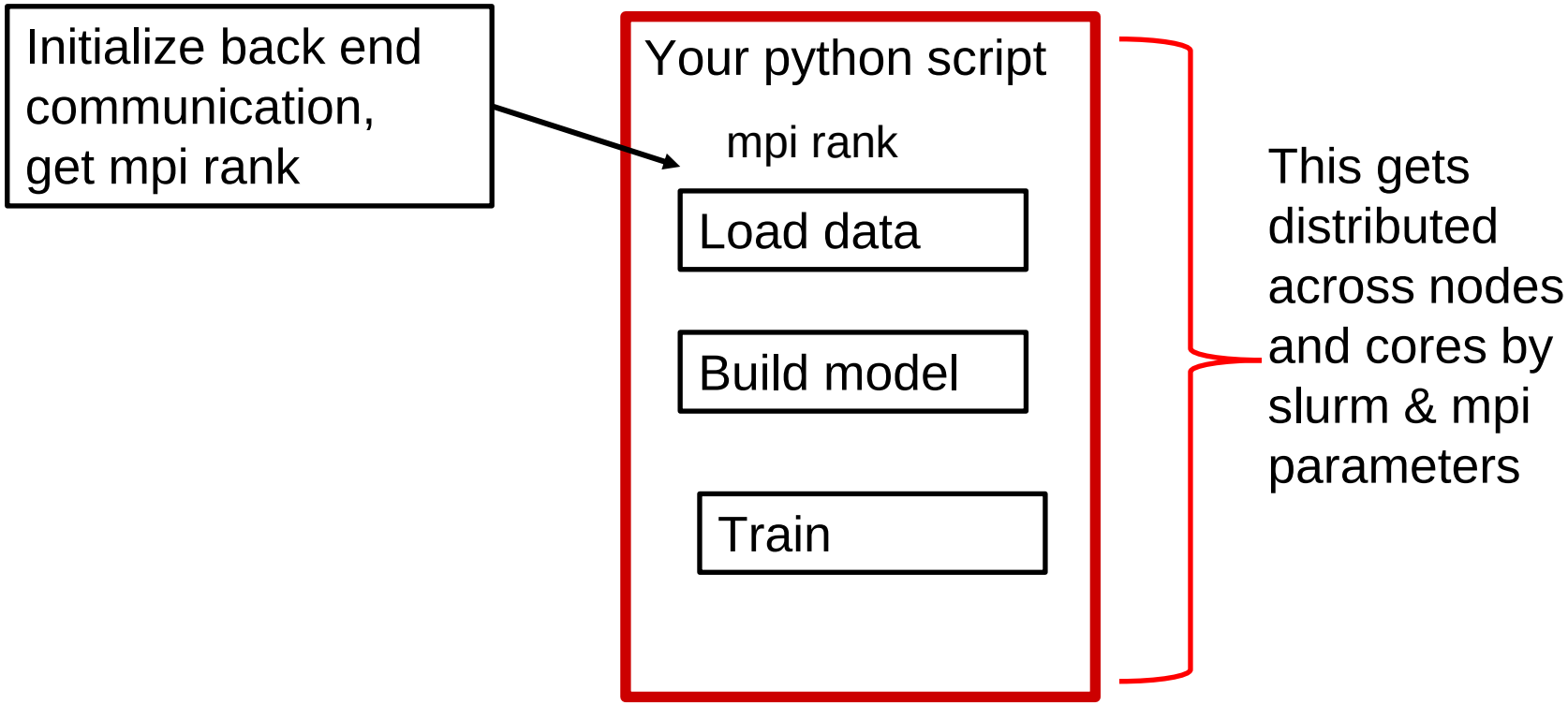
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

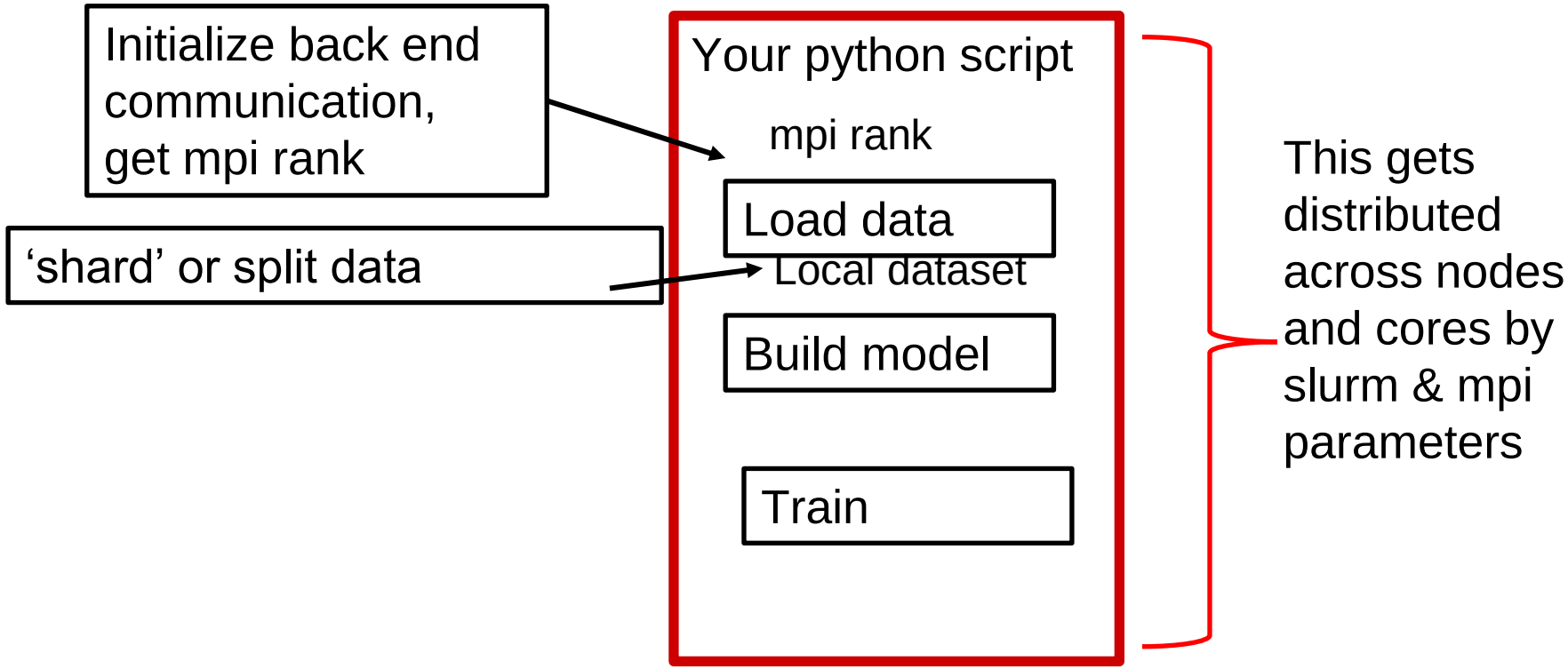
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

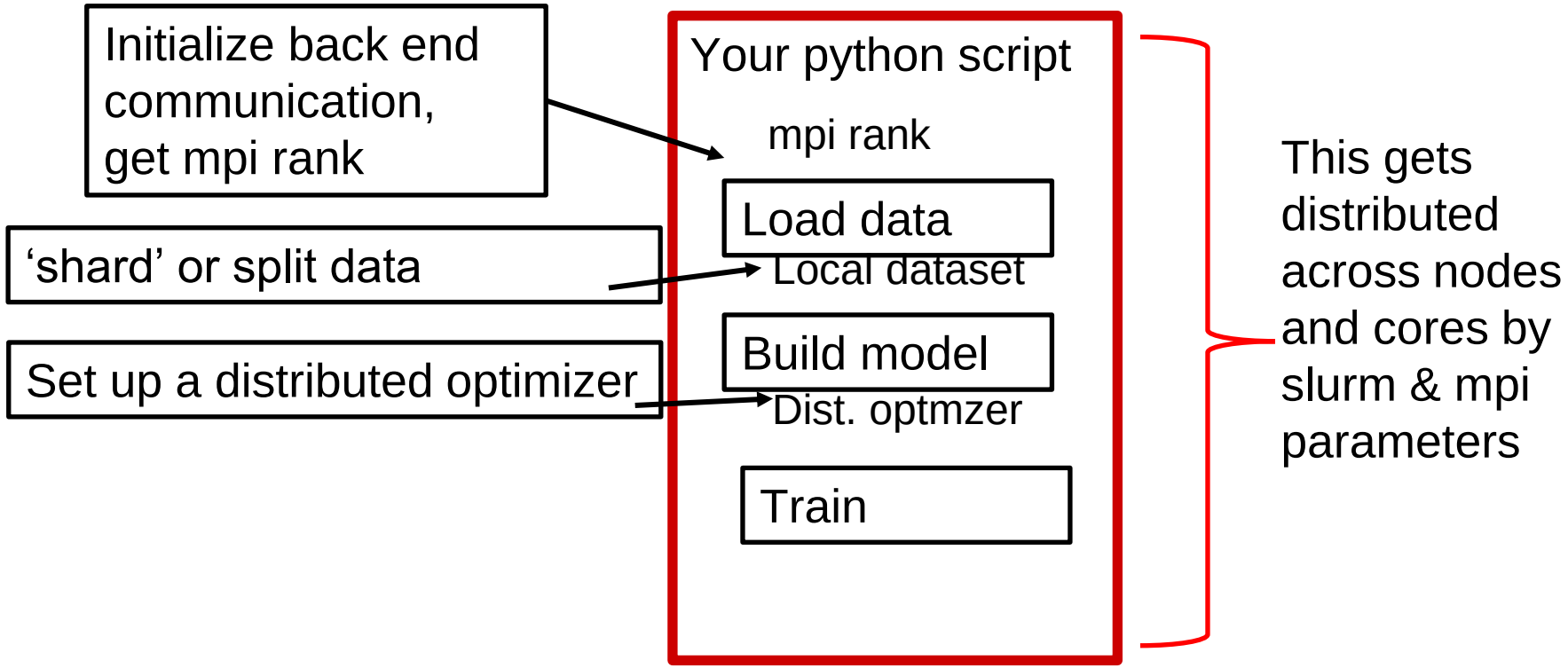
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

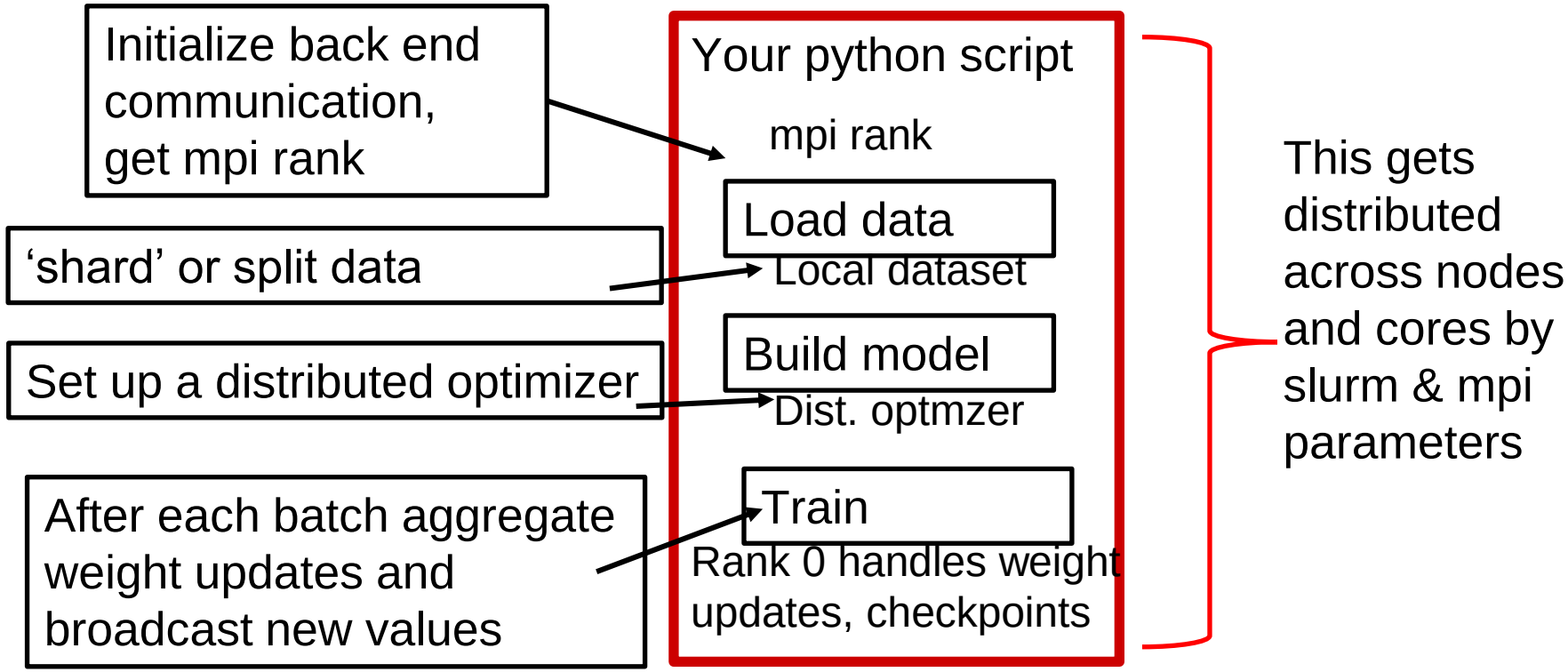
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

mpirun –n **number of tasks** singularity → python



mpi launches one instance per processor

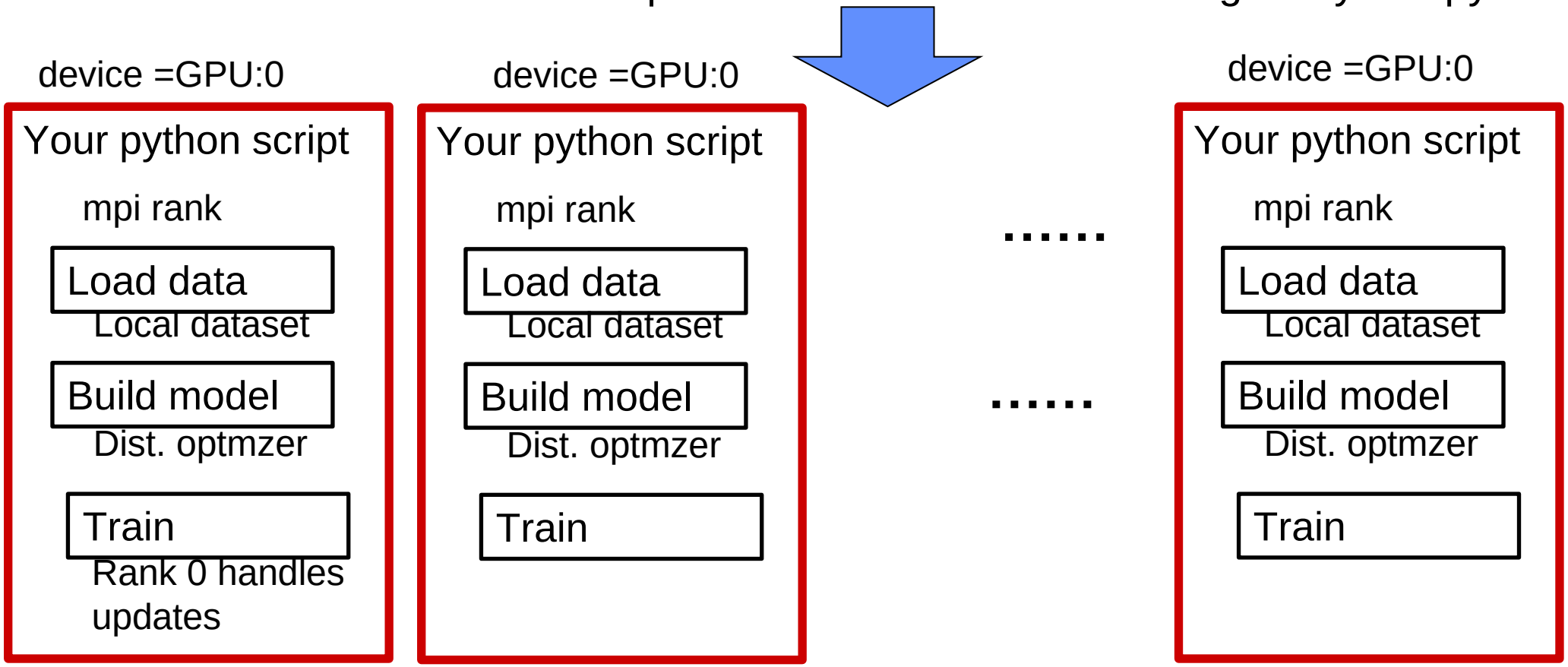
In slurm batch script:

mpirun –n **number of tasks** singularity → python

mpi launches one instance per processor

In slurm batch script:

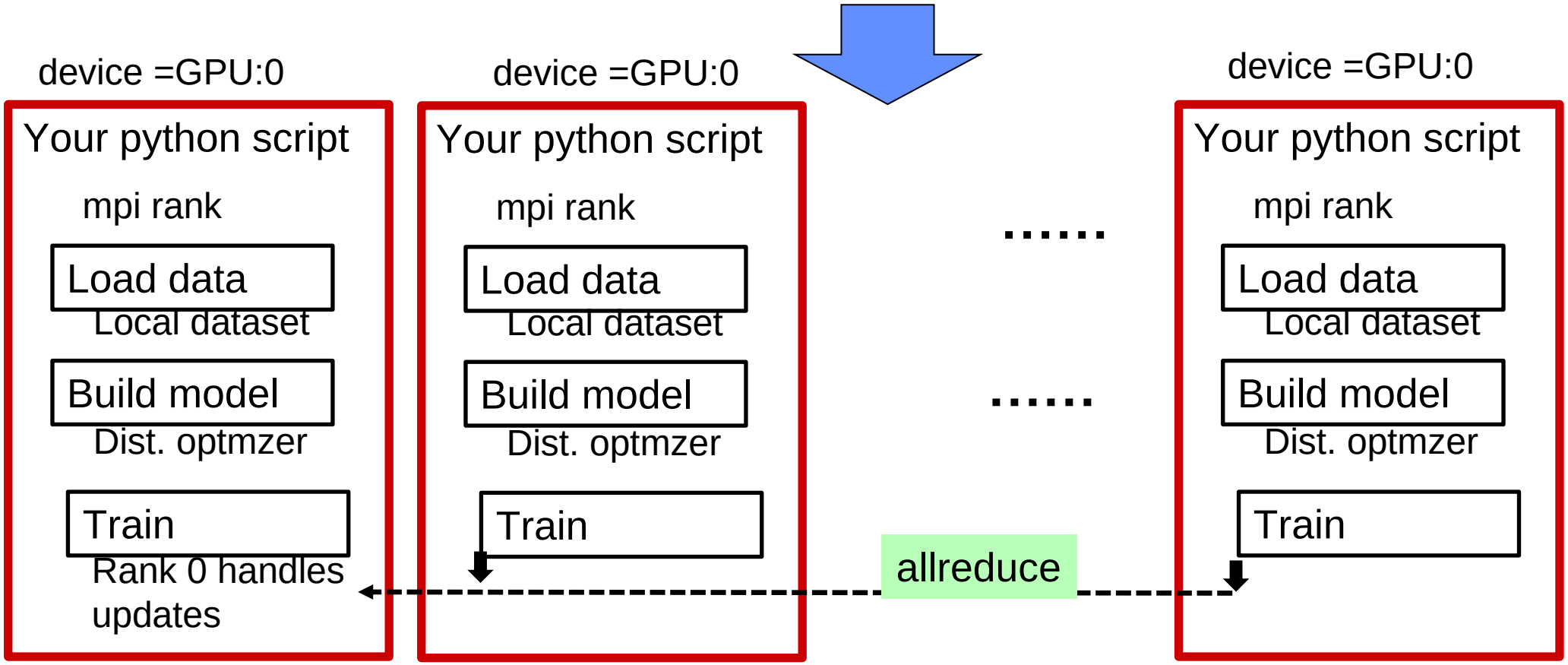
mpirun –n **number of tasks** singularity → python



For each input batch: Horovod will aggregate & share weights updates

In slurm batch script:

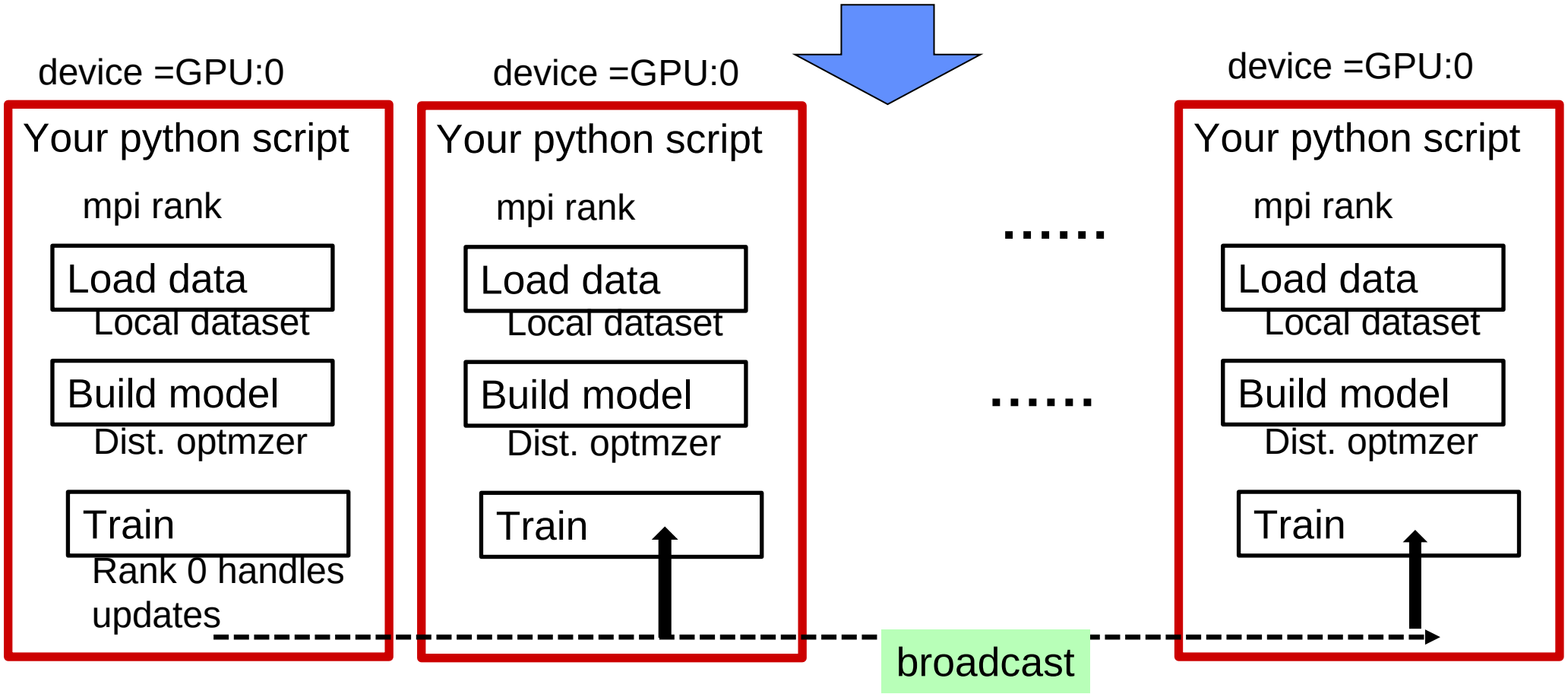
mpirun –n **number of tasks** singularity → python



For each input batch: Horovod will aggregate & share weights updates

In slurm batch script:

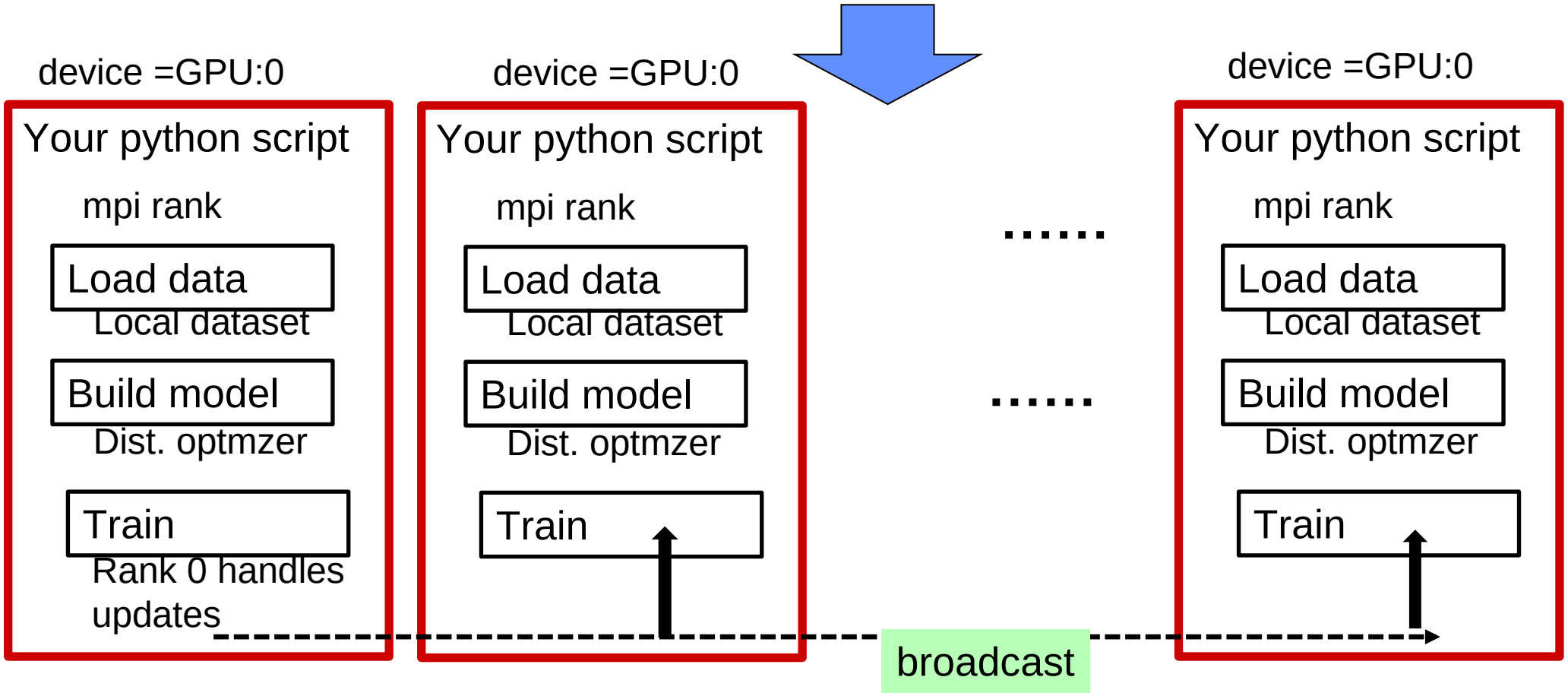
mpirun –n **number of tasks** singularity → python



For each input batch: Horovod will aggregate & share weights updates

In slurm batch script:

mpirun –n **number of tasks** singularity → python



Bigger batch size helps, but it uses more memory

Code snippets – Horovod functions

Not many lines of code, but becareful with sharding, batch size,
See https://horovod.readthedocs.io/en/latest/keras.html

Initialize back end communication, get mpi rank

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

'shard' or split data

```
Note: many ways to do this, either directly splitting numpy
arrays and/or using TF datasets
```

Set up a distributed optimizer

```
optimizer2use  = hvd.DistributedOptimizer(……)
```

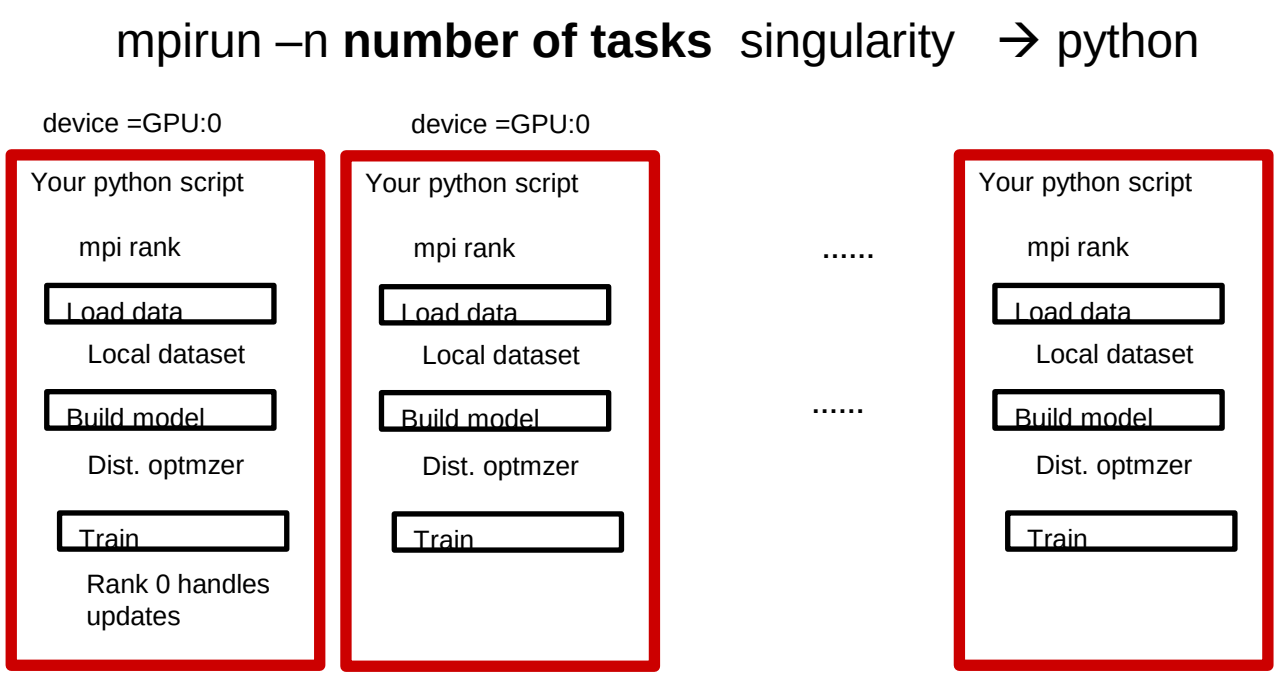
After each batch aggregate weight updates and broadcast new values

```
model.compile(optimizer    =optimizer2use,
 …
experimental_run_tf_function=False)
```

Exercise, multinode MNIST programming and execution

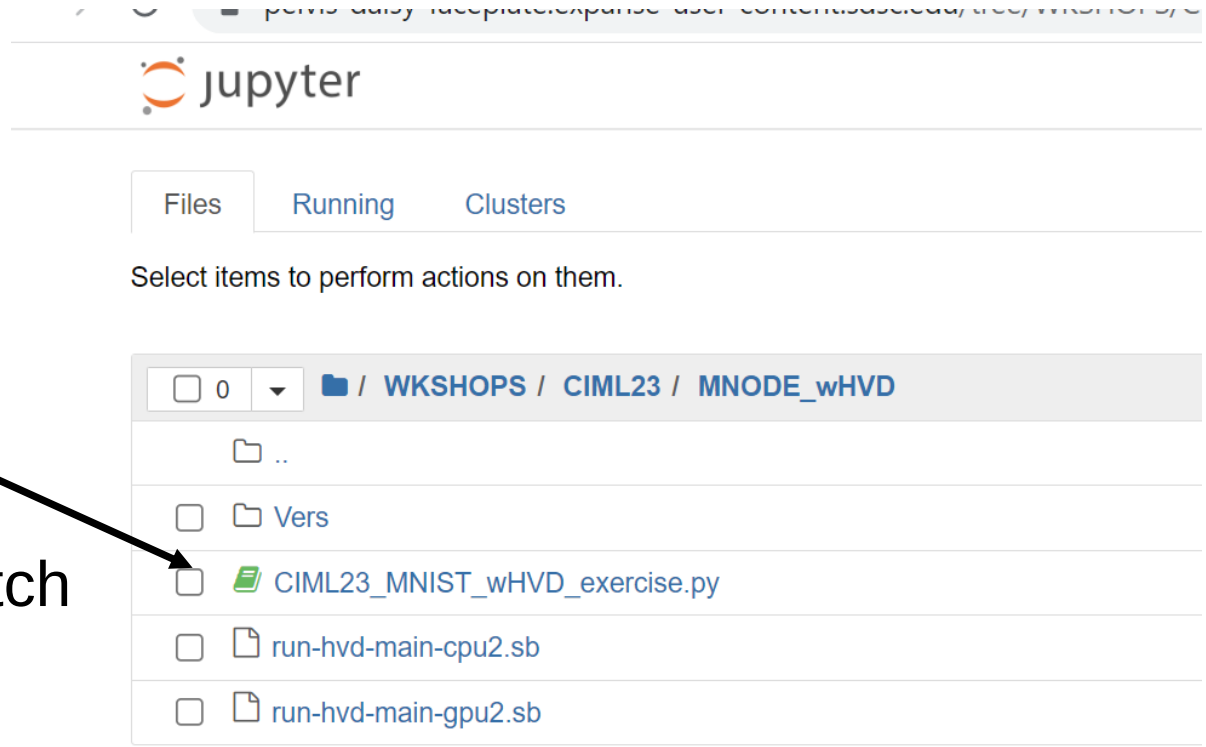
- **Goal: Get familiar with Keras and Horovod coding for multinode execution**
- **Goal: Get familiar with slurm batch script multinode parameters**
- **Let's login and start a notebook**

(see next pages for quick overview)



In terminal]$ **jupyter-compute-tensorflow**

In jupyter notebook session open the *MNIST_wHVD_exercise* notebook



Open the *run-hvd-main-cpu2.sb* slurm batch script

Follow instructions in the notebook

Code highlights

Initialize back end communication, get mpi rank

```
#-----------------------------------------------
import horovod.keras as hvd
hvd.init()
print('INFO, global rank:',hvd.rank(), ' localrank ',hvd.local_rank())
#-------------------------------------------------------------
```

split data using numpy arrays

```
# ------------------ Get Dataset ---------------------------
per_worker_batch_size = 32        #Pick factors of 32 (especially for GPU)
num_workers           = hvd.size()
```

Set up a distributed optimizer to manage weight updates

```
# -------------------------------------------------------

#------- Enter the num of processes to scale the learning rate here -------
optimizer2use    = tf.keras.optimizers.Adam(learning_rate=0.001*hvd.size())  #<<<<<<---------
optimizer2use    = hvd.DistributedOptimizer(optimizer2use)

# ------------ for HVD ---------------------------------------
#Specify `experimental_run_tf_function=False` to ensure TensorFlow
```

Also run sbatch command for this slurm script

Try reviewing stdout output file

jupyter run-hvd-main-cpu2.sb 15 minutes ago

Logout

File Edit View Language Plain Text

```
#SBATCH --job-name=tfhvd-cpu
#SBATCH --account=use300
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16  #<<<<<------ change this to 16 and observe changes in training time
#SBATCH --cpus-per-task=1
#SBATCH --mem=243G
#SBATCH --time=00:15:00
#SBATCH --output=slurm.cpu2.%x.o%j.out
```

```
p4rodrig@login01 MNODE_wHVD]$
p4rodrig@login01 MNODE_wHVD]$ grep 'rank' stdout_cpu2_mnist_32.txt |grep ' 0'
NFO, global rank: 0  localrank  0
NFO, global rank: 16  localrank  0
NFO,cpus available rank: 0   [PhysicalDevice(name='/physical_device:CPU:0', dev
ce_type='CPU')]
This is rank 0 instance model evaluation
p4rodrig@login01 MNODE_wHVD]$
```

```
[p4rodrig@login01 MNODE_wHVD]$
[p4rodrig@login01 MNODE_wHVD]$ grep 'done, rk: 15' stdout_*
stdout_cpu2_mnist_32.txt:INFO,done, rk: 15  train time: 2.48225  secs
stdout_mainhvd_cpu2.txt:INFO,done, rk: 15  train time: 2.31222  secs
[p4rodrig@login01 MNODE_wHVD]$
[p4rodrig@login01 MNODE_wHVD]$
```

optional extra to try

You can run:

$ squeue –u userid to see nodes running your job

$ ssh exp-XX-YY to login to node

$ top –u userid to see processing on a CPU job

```
p4rodrig@exp-1-35:~
top - 14:40:05 up 315 days,  2:34,  1 user,  load average: 78.65, 76.25, 76.07
Threads: 2499 total,  90 running, 2309 sleeping,   0 stopped, 100 zombie
%Cpu(s): 69.0 us,  0.8 sy,  0.0 ni, 29.8 id,  0.0 wa,  0.4 hi,  0.0 si,  0.0 st
MiB Mem : 257509.7 total, 142734.5 free,  40306.1 used,  74469.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used. 213120.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND                P
2460166 p4rodrig  20   0 2967760 362936 181296 R  99.4   0.1   0:11.58 python3             43
2460163 p4rodrig  20   0 2967760 363028 181380 R  99.2   0.1   0:11.15 python3             46
2460161 p4rodrig  20   0 2967764 363072 181420 R  99.0   0.1   0:11.24 python3             34
2460153 p4rodrig  20   0 2967760 363688 181204 R  98.5   0.1   0:12.33 python3             35
2460167 p4rodrig  20   0 2967760 362904 181260 R  98.3   0.1   0:11.55 python3             47
2460155 p4rodrig  20   0 2967744 362908 181280 R  98.1   0.1   0:11.92 python3             39
2460158 p4rodrig  20   0 2967760 362880 181240 R  97.9   0.1   0:10.39 python3             32
2460164 p4rodrig  20   0 2967764 363004 181360 R  97.9   0.1   0:10.01 python3             45
2460162 p4rodrig  20   0 2541772 363332 181692 R  97.9   0.1   0:11.78 python3             38
2460154 p4rodrig  20   0 2967760 362992 181352 R  97.3   0.1   0:11.32 python3             44
2460156 p4rodrig  20   0 2967756 362972 181332 R  92.8   0.1   0:08.37 python3             42
2460160 p4rodrig  20   0 2902224 363032 181388 R  89.1   0.1   0:11.38 python3             36
2460169 p4rodrig  20   0 2967760 362868 181224 S  82.9   0.1   0:10.25 python3             33
2460157 p4rodrig  20   0 2967760 362912 181268 R  76.6   0.1   0:10.16 python3             40
2460159 p4rodrig  20   0 2967760 362932 181292 R  75.4   0.1   0:08.42 python3             37
2460171 p4rodrig  20   0 2967760 362844 181204 S  63.4   0.1   0:10.22 python3             45
2460428 p4rodrig  20   0   67360   7088   3484 R   1.0   0.0   0:00.26 top                122
2458820 p4rodrig  20   0   89628   9512   7896 S   0.0   0.0   0:00.06 systemd             55
2458836 p4rodrig  20   0  326556  12812      0 S   0.0   0.0   0:00.00 (sd-pam)           122
```

Or run: $ nvidia-smi to see usage on GPU devices

```
p4rodrig@login02:/expanse/lustre/projects/sds164/p4rodrig/TFwHVDtests
[p4rodrig@login02 TFwHVDtests]$ squeue -u p4rodrig
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
          21782434       gpu tfhvd-gp p4rodrig  R       0:01      1 exp-10-57
[p4rodrig@login02 TFwHVDtests]$ ssh exp-10-57
[p4rodrig@exp-10-57 ~]$ nvidia-smi
Wed Apr 19 17:50:57 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.39.01    Driver Version: 510.39.01    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:18:00.0 Off |                    0 |
| N/A   37C    P0    55W / 300W |   1005MiB / 32768MiB |     11%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:3B:00.0 Off |                    0 |
| N/A   37C    P0    55W / 300W |   1005MiB / 32768MiB |      6%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

- **End**