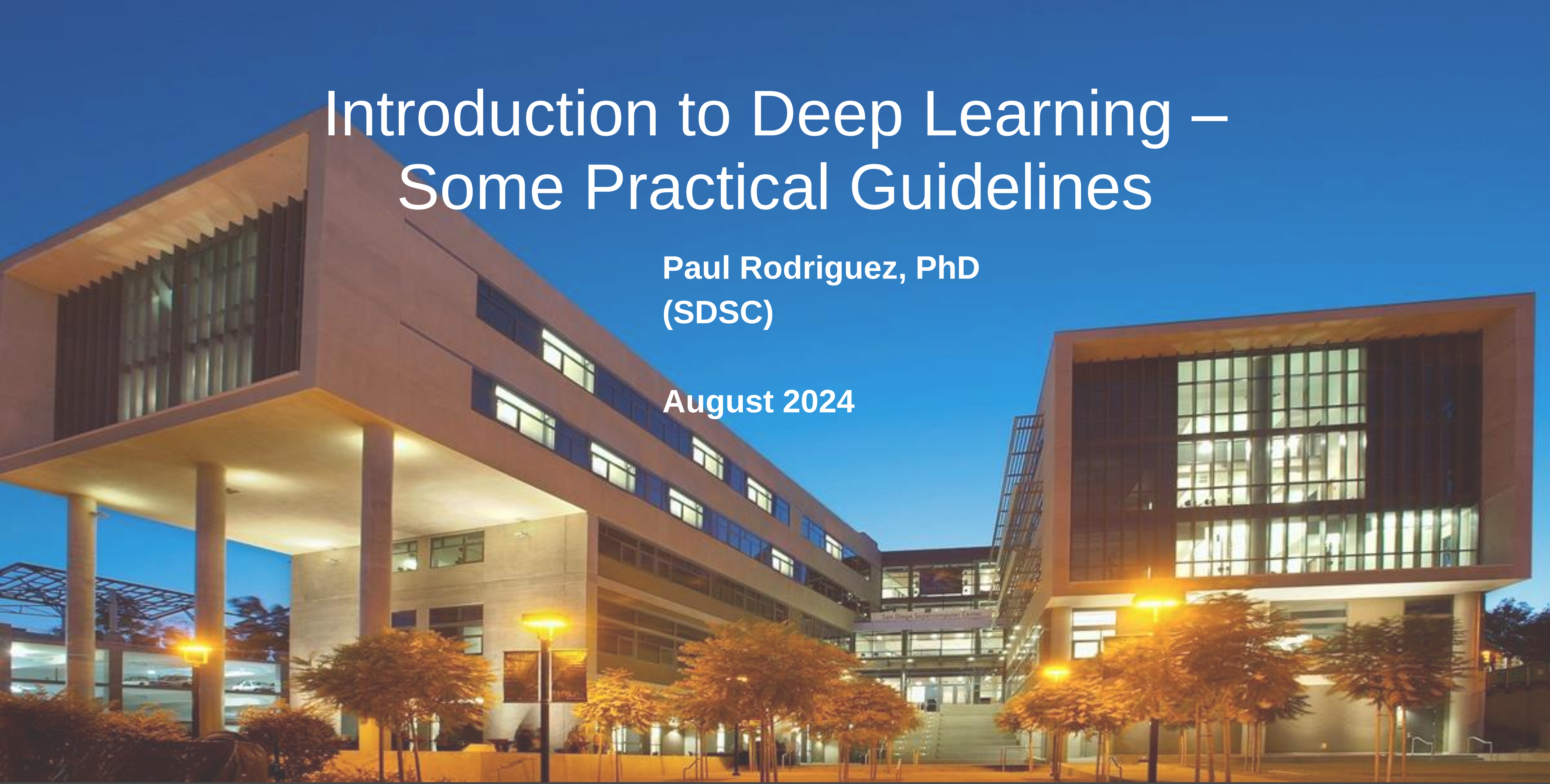


Introduction to Deep Learning – Some Practical Guidelines

Paul Rodriguez, PhD
(SDSC)

August 2024

Outline

- **Part II - 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

Parallelizing 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:
  - e.g. the loss function, using 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

  (Raytune is similar tool for Pytorch)

Keras Hyperparameter Search Tool

- Keras Hypertuner class implements several search strategies:

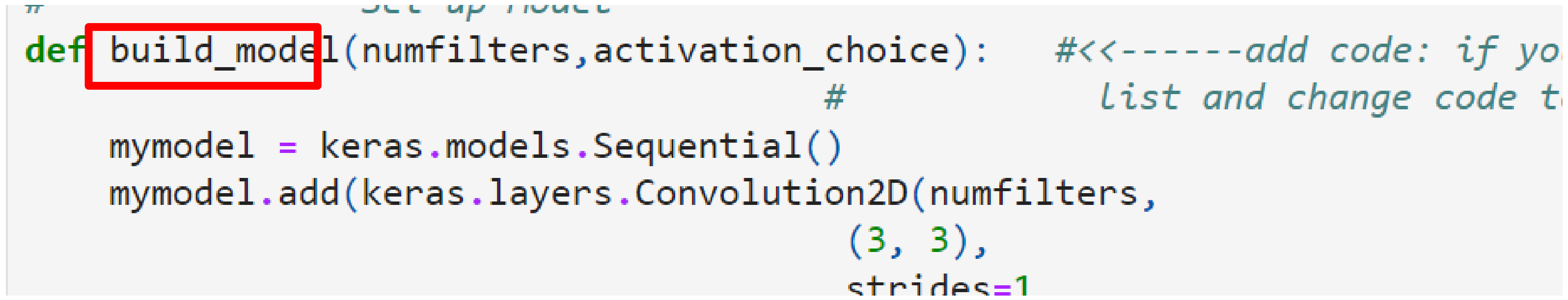
Hyperband is like a tournament of hyperparameter configurations

RandomSearch will search randomly through the space of configurations

Bayesian optimization is like a function approximation to pick out next configuration

Keras Tuner code snippet

Set up function to make the model

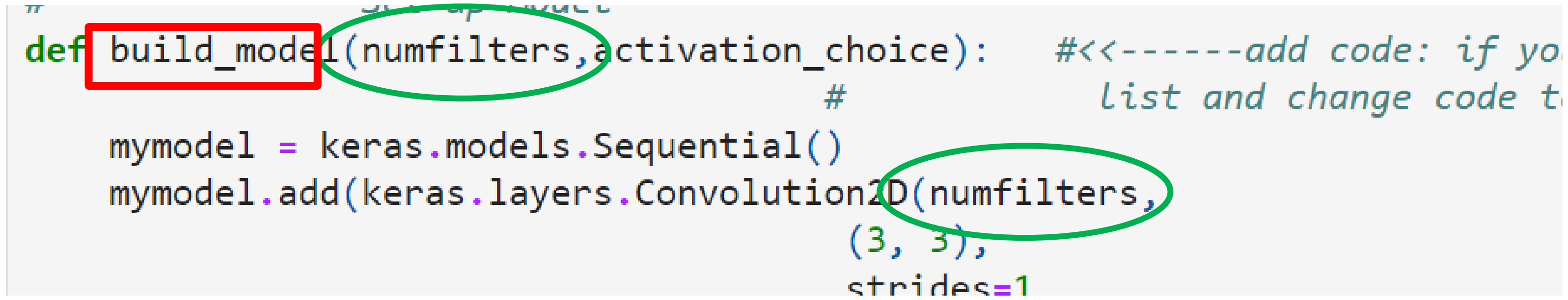


```
def build_model(numfilters,activation_choice):    #<<------add code: if yo
                                                #           list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                                 (3, 3),
                                                 strides=1
```

Keras Tuner code snippet –

Set up function to make the model

Set up hyperparameter choices

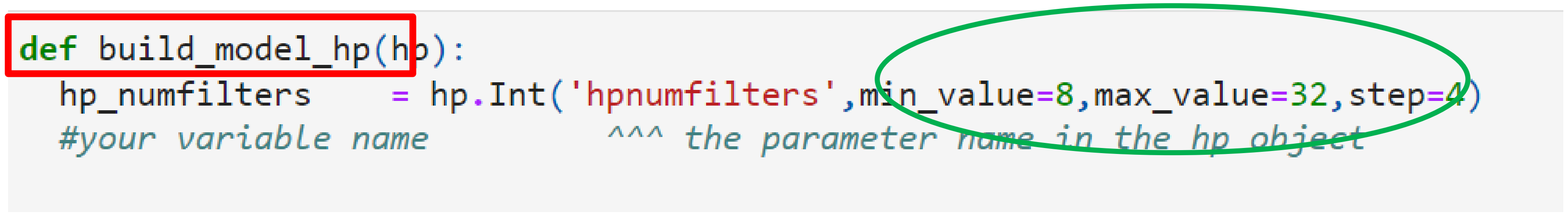


```
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                                 #           list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                           (3, 3),
                                           strides=1
```

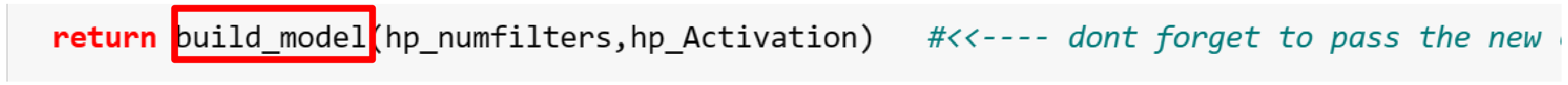
Keras Tuner code snippet – put build-model inside a ‘wrapper’ function

Set up function to make the model

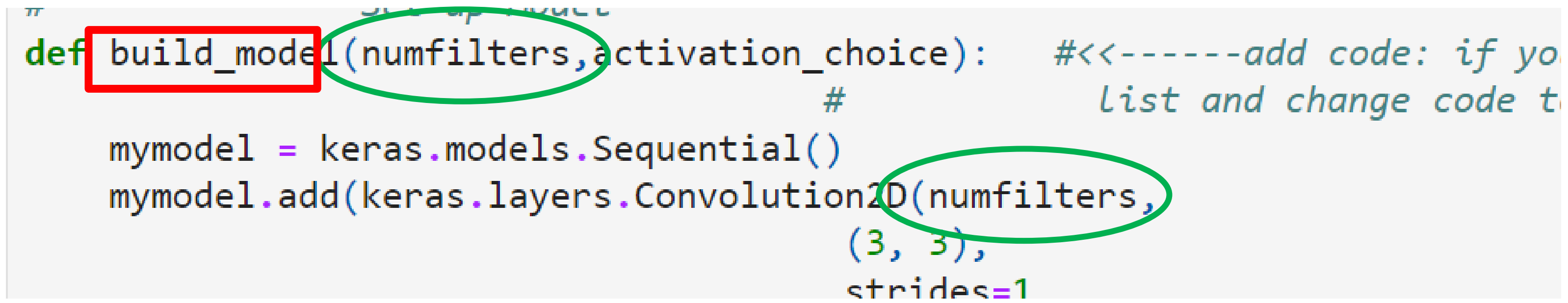
Set up hyperparameter choices



```
def build_model_hp(hp):
  hp_numfilters     = hp.Int('hpnumfilters',min_value=8,max_value=32,step=4)
  #your variable name          ^^^ the parameter name in the hp object
```



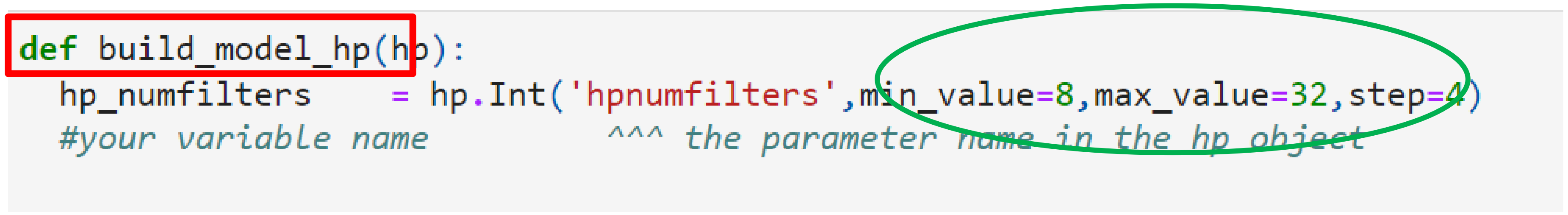
```
  return build_model(hp_numfilters,hp_Activation)   #<<---- dont forget to pass the new
```



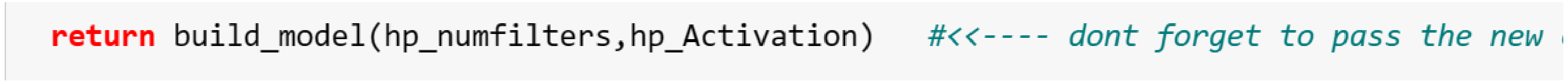
```
#            Set up model
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                                #          list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                           (3, 3),
                                           strides=1
```

Keras Tuner code snippet – put build-model inside a ‘wrapper’ function

Set up function to make the model

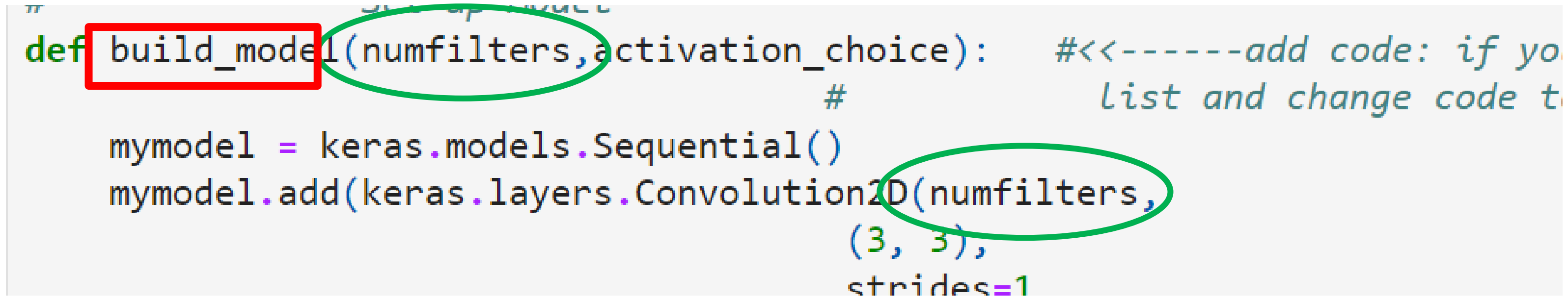


```
def build_model_hp(hp):
    hp_numfilters     = hp.Int('hpnumfilters',min_value=8,max_value=32,step=4)
    #your variable name          ^^^ the parameter name in the hp object
```



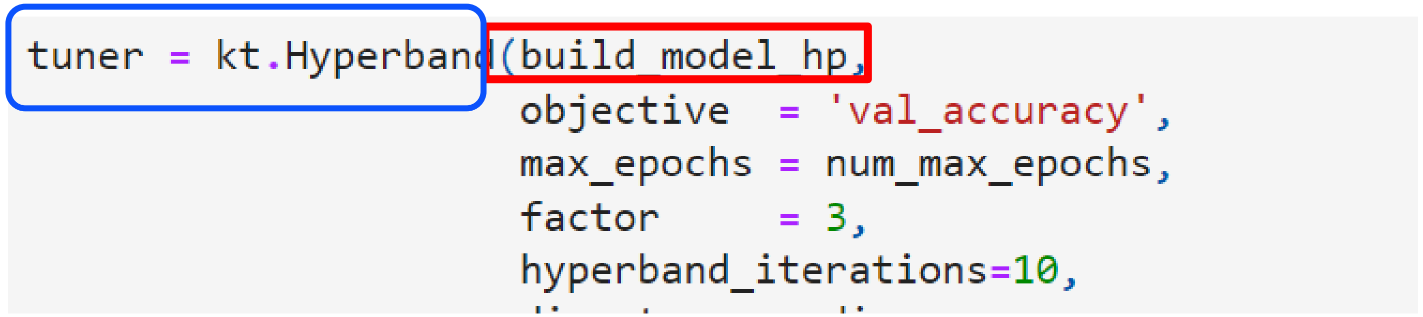
```
    return build_model(hp_numfilters,hp_Activation)   #<<---- dont forget to pass the new
```

Set up hyperparameter choices



```
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                      #           list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                      (3, 3),
                                      strides=1
```

Define ‘tuner’ object: use the wrapper and model fit to search configurations



```
tuner = kt.Hyperband(build_model_hp,
                     objective    = 'val_accuracy',
                     max_epochs = num_max_epochs,
                     factor       = 3,
                     hyperband_iterations=10,
```

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)

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 a model runs for 1 configuration of hyperparameters**
- **Organize jobs into reasonable chunks of work**
- **For large models consider model-checkpoints**
- **Tensorboard is available on Expanse (ask for details to run securely)**

note on using GPU

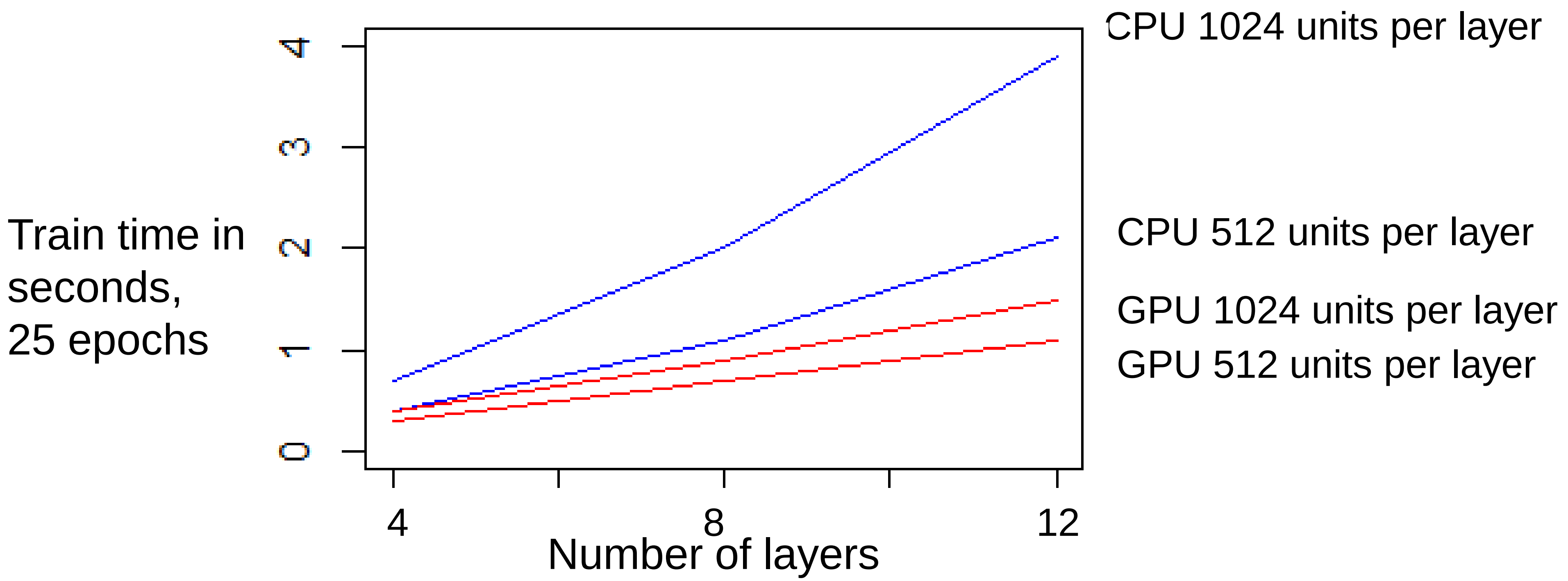
- **GPU node has multiple GPU devices**

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

Code snippet to check for GPU devices

```
import tensorflow as tf

gpu_devices = tf.config.list_physical_devices("GPU")
```

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

Parallelism in Deep Learning

- Two Goals

1 Speed Up Learning - as data scales up training takes longer

2 Optimize Memory - as models scale up they take up too much memory

e.g. V100s have 32Gb limit and 8B float32 parameters would fill that

Parallelism strategies

- Data Parallelism: partition data and copy the model across devices,
  (this is probably easiest thing to do, least programming)

- Pipeline Parallelism: split up the model across devices, i.e. inter-layer
  (you organize layers)

- Tensor Parallelism: split up weight matrix across devices, i.e. intra-layer
  (model has to support it)

Reducing memory footprint

- Using mixed precision (e.g. bfloat16 for weights) lowers memory at a cost of accuracy

- ‘DEEPSPEED’ package partitions the optimization calculations at a cost of communication

Parallel DL models with multiple nodes/devices

- **Data Parallel:**

  Each device trains a copy of the model with a part of the data
  Weight updates have to be aggregated across devices

- **Main tools on Expanse:**

  Keras/Tensorflow ‘strategy’ (simple for multidevice on 1 node)

  or

  Horovod MPI wrappers (for multinode)

Keras/Tensorflow strategy single GPU node

1. Get a list of gpu devices available

```
gpus_list  = tf.config.experimental.list_physical_devices('GPU’)
```

2. Set up a ‘mirror’ strategy

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

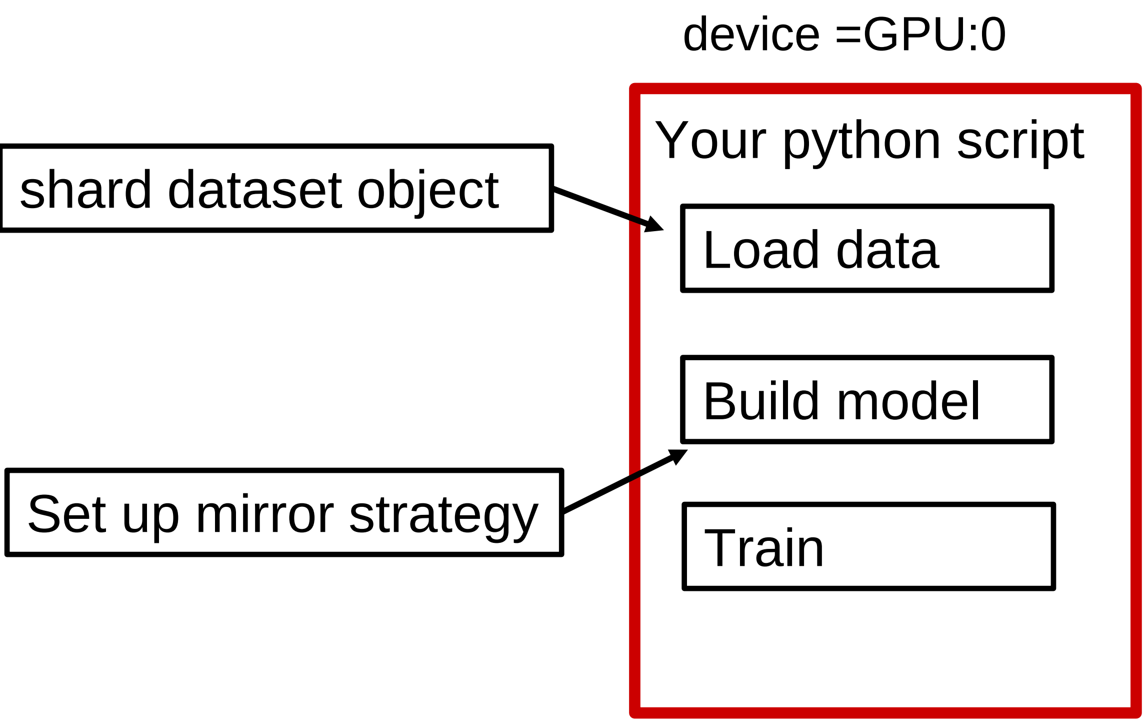
3. Use strategy scope around the model definition (so model gets copied)

```
with mirrored_strategy.scope():
      multi_dev_model=build_model()
```

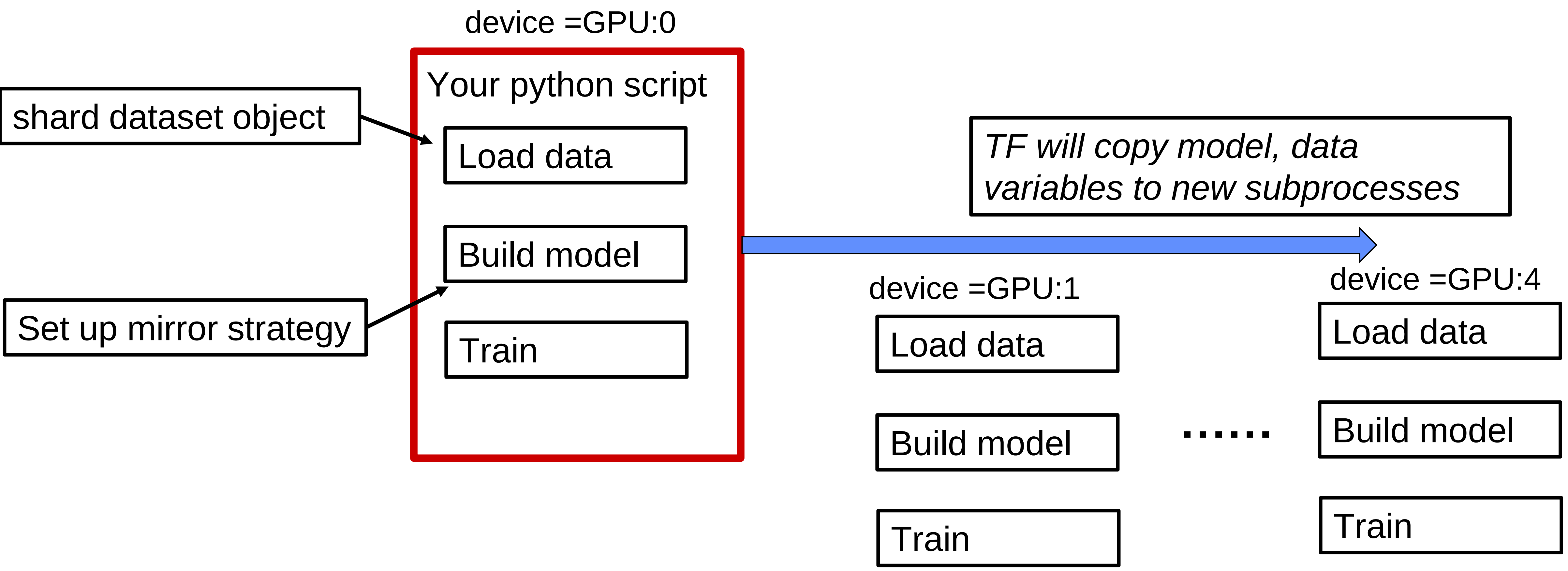
4. A tensorflow dataset object will ‘shard’ (ie split up) data among GPU devices

```
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, Y_train)) …
```

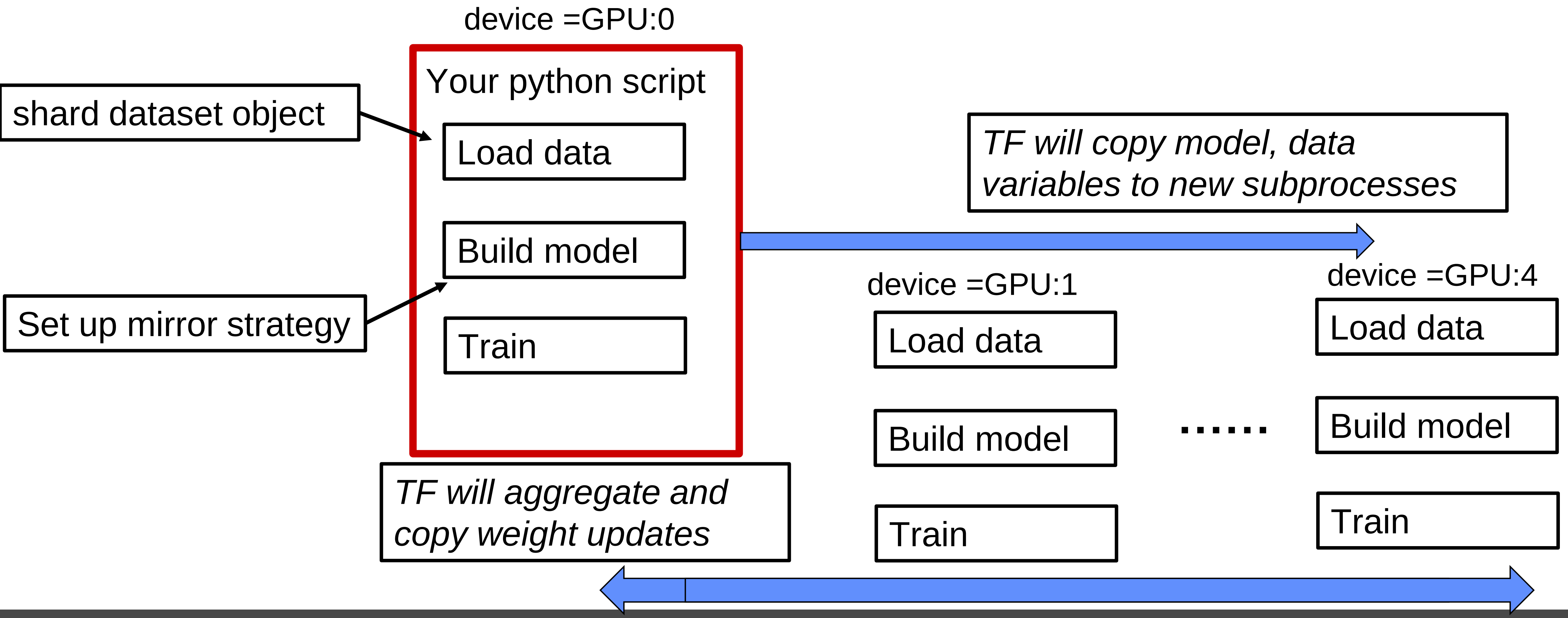
Mirror Strategy single GPU node



Mirror Strategy single GPU node



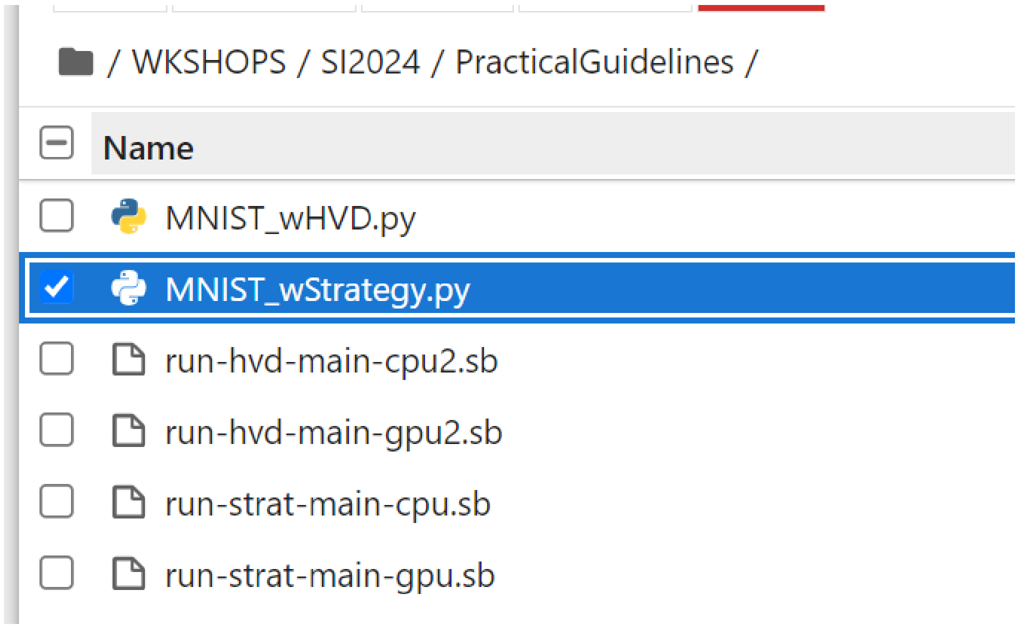
Mirror Strategy single GPU node



Keras/Tensorflow strategy single GPU node

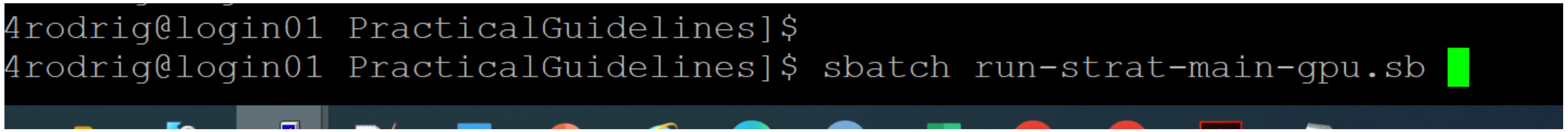
Example, in github repo find:

MNIST_wStrategy.py
run-strat-main-gpu.sb
run-strat-main-cpu.sb


/ WKSHOPS / SI2024 / PracticalGuidelines /

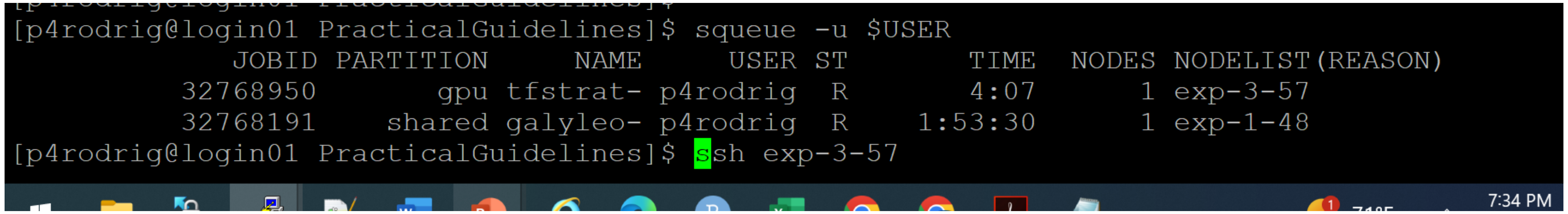
	Name
☐	MNIST_wHVD.py
✓	MNIST_wStrategy.py
☐	run-hvd-main-cpu2.sb
☐	run-hvd-main-gpu2.sb
☐	run-strat-main-cpu.sb
☐	run-strat-main-gpu.sb

From terminal window submit a batch job:


```
4rodrig@login01 PracticalGuidelines]$
4rodrig@login01 PracticalGuidelines]$ sbatch run-strat-main-gpu.sb
```

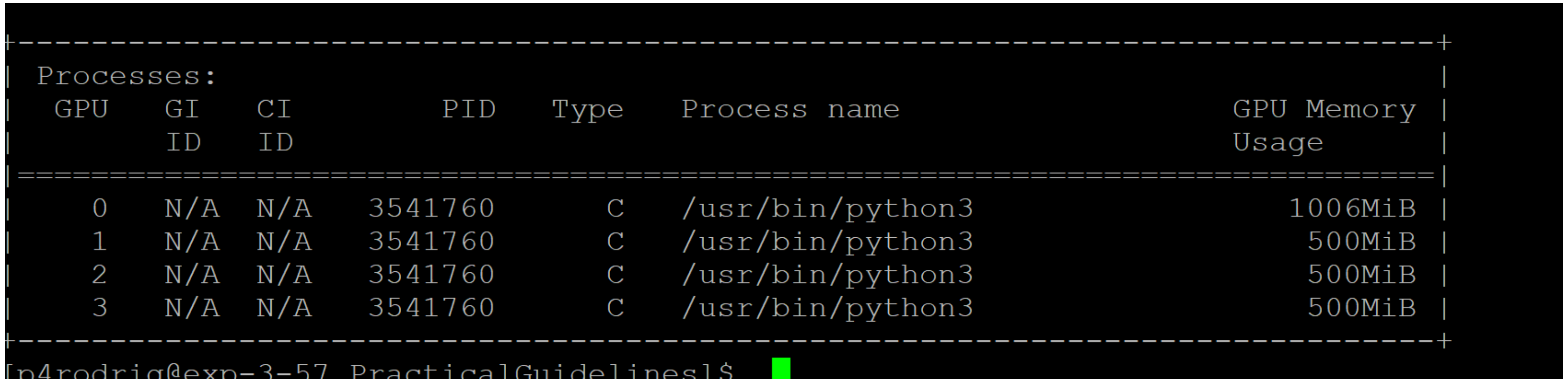
Keras/Tensorflow strategy single GPU node

View your job queue and ‘ssh’ into the GPU node



```
[p4rodrig@login01 PracticalGuidelines]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          32768950       gpu tfstrat- p4rodrig  R       4:07      1 exp-3-57
          32768191    shared galyleo- p4rodrig  R    1:53:30      1 exp-1-48
[p4rodrig@login01 PracticalGuidelines]$ ssh exp-3-57
```

In the GPU node run ‘nvidia-smi’ to view GPU usage

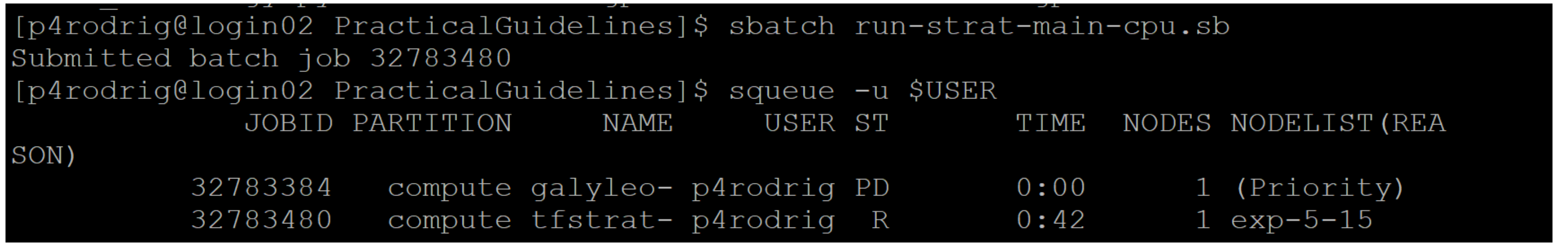


```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   3541760      C   /usr/bin/python3                 1006MiB |
|    1   N/A  N/A   3541760      C   /usr/bin/python3                  500MiB |
|    2   N/A  N/A   3541760      C   /usr/bin/python3                  500MiB |
|    3   N/A  N/A   3541760      C   /usr/bin/python3                  500MiB |
+-----------------------------------------------------------------------------+
[p4rodrig@exp-3-57 PracticalGuidelines]$
```

TF has copied the model into each GPU, sharded the data, and coordinated learning

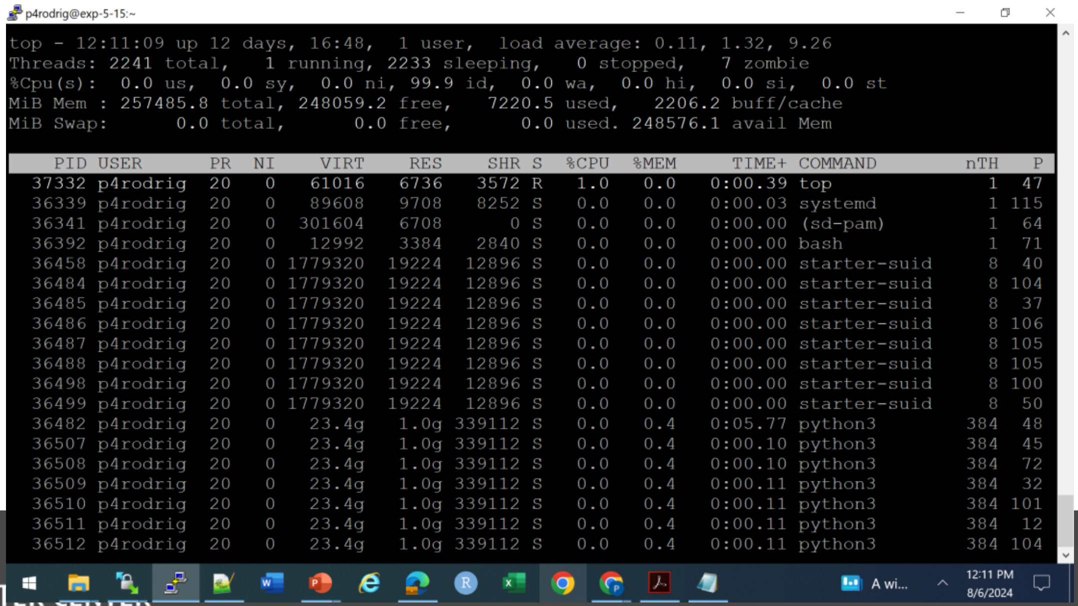
Keras/Tensorflow strategy single CPU node

View your job queue and 'ssh' into the CPU node



```
[p4rodrig@login02 PracticalGuidelines]$ sbatch run-strat-main-cpu.sb
Submitted batch job 32783480
[p4rodrig@login02 PracticalGuidelines]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
          32783384   compute galyleo- p4rodrig PD       0:00      1 (Priority)
          32783480   compute tfstrat- p4rodrig  R       0:42      1 exp-5-15
```

In the CPU node run top –u $USER
Then 'H' , 'f', down arrow to 'n-th' , space bar to toggle, (repeat for P), esc



```
top - 12:11:09 up 12 days, 16:48,  1 user,  load average: 0.11, 1.32, 9.26
Threads: 2241 total,   1 running, 2233 sleeping,   0 stopped,   7 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 257485.8 total, 248059.2 free,   7220.5 used,   2206.2 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used. 248576.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND          nTH   P
  37332 p4rodrig  20   0   61016   6736   3572 R   1.0   0.0   0:00.39 top                1  47
  36339 p4rodrig  20   0   89608   9708   8252 S   0.0   0.0   0:00.03 systemd            1 115
  36341 p4rodrig  20   0  301604   6708      0 S   0.0   0.0   0:00.00 (sd-pam)           1  64
  36392 p4rodrig  20   0   12992   3384   2840 S   0.0   0.0   0:00.00 bash               1  71
  36458 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8  40
  36484 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8 104
  36485 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8  37
  36486 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8 106
  36487 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8 105
  36488 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8 105
  36498 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8 100
  36499 p4rodrig  20   0 1779320  19224  12896 S   0.0   0.0   0:00.00 starter-suid       8  50
  36482 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:05.77 python3          384  48
  36507 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.10 python3          384  45
  36508 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.10 python3          384  72
  36509 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.11 python3          384  32
  36510 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.11 python3          384 101
  36511 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.11 python3          384  12
  36512 p4rodrig  20   0   23.4g   1.0g 339112 S   0.0   0.4   0:00.11 python3          384 104
```

TF uses threads and available CPU cores for parallelization

Keras/Tensorflow strategy multiple GPU/CPU node

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

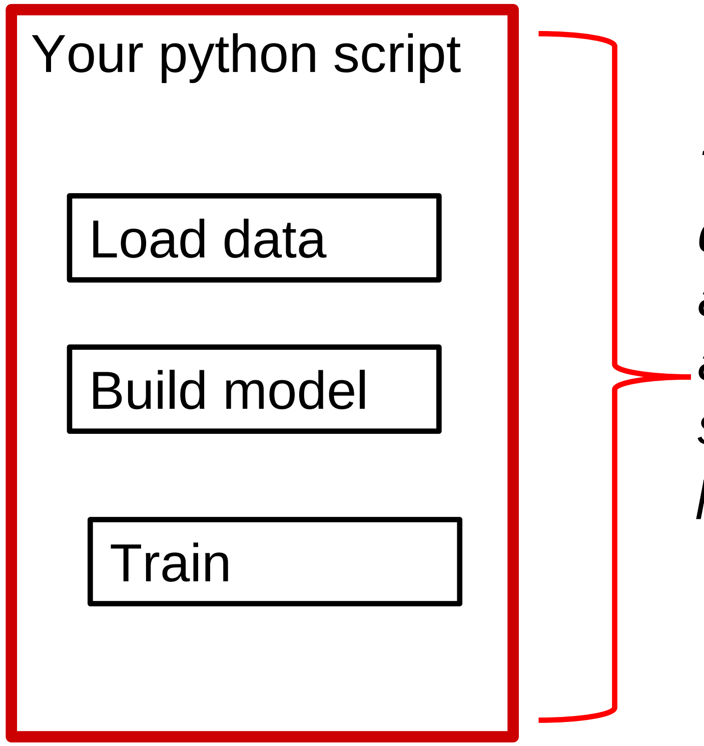
But, on HPC systems resources are shared so IP addresses are dynamic

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

Multinode, mpi launches instances

In slurm batch script:

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

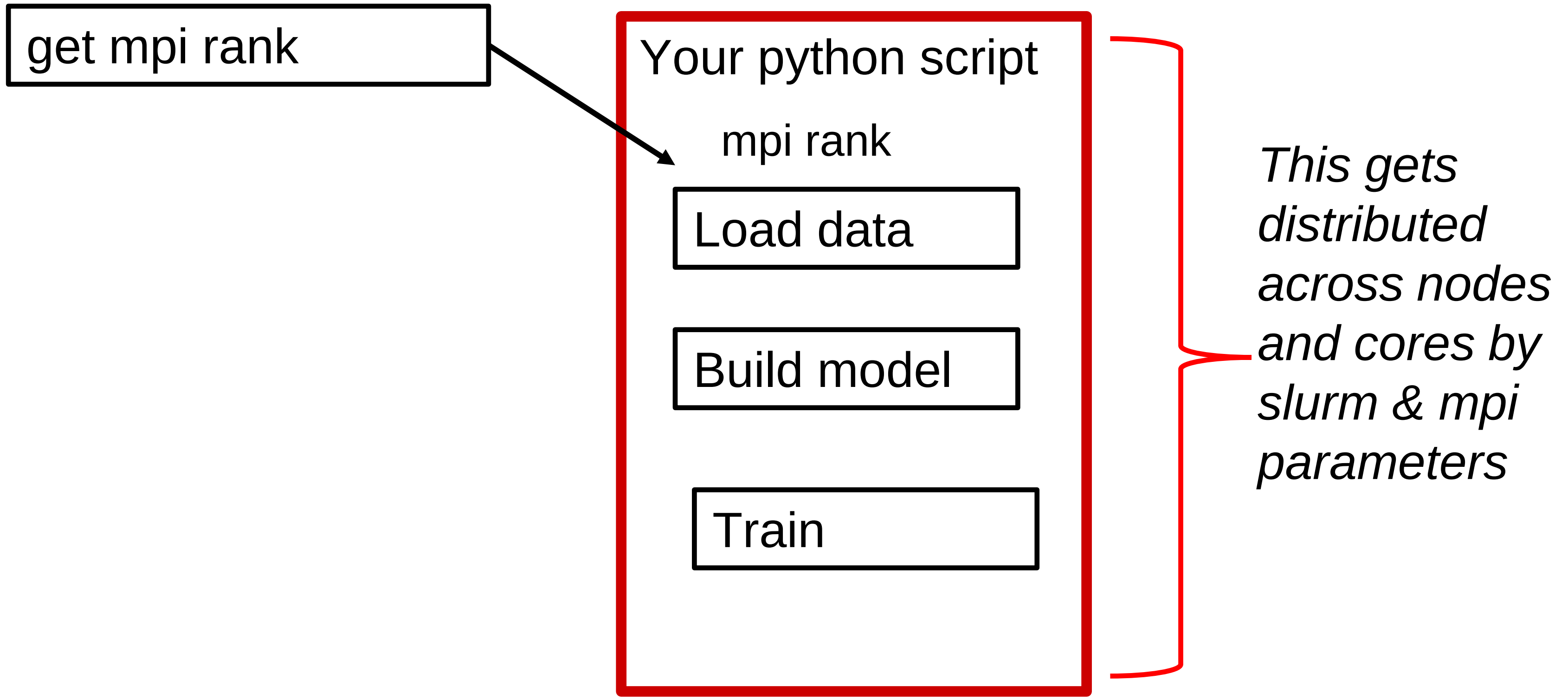


This gets distributed across nodes and cores by slurm & mpi parameters

Multinode, mpi launches instances

In slurm batch script:

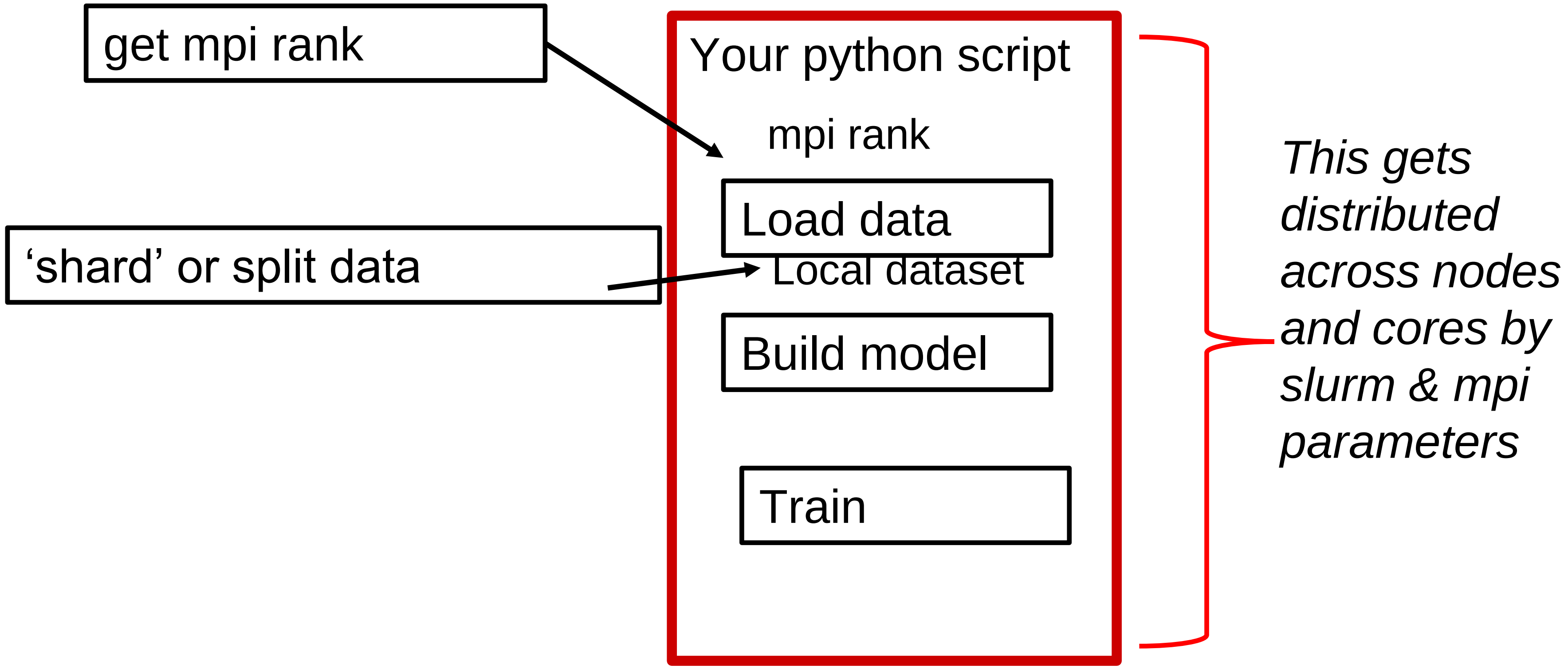
mpirun –n **number of tasks** singularity → python


get mpi rank
Your python script
mpi rank
Load data
Build model
Train
This gets distributed across nodes and cores by slurm & mpi parameters

Multinode, mpi launches instances

In slurm batch script:

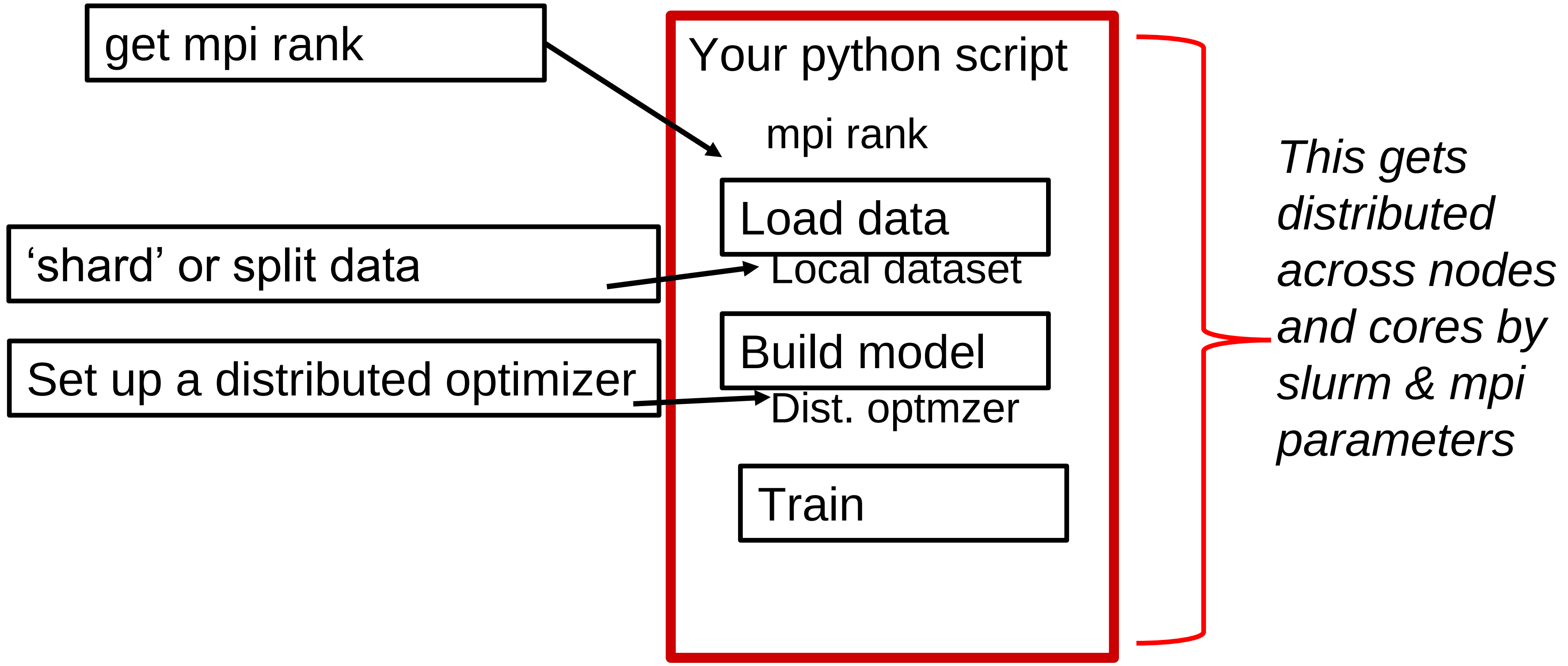
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

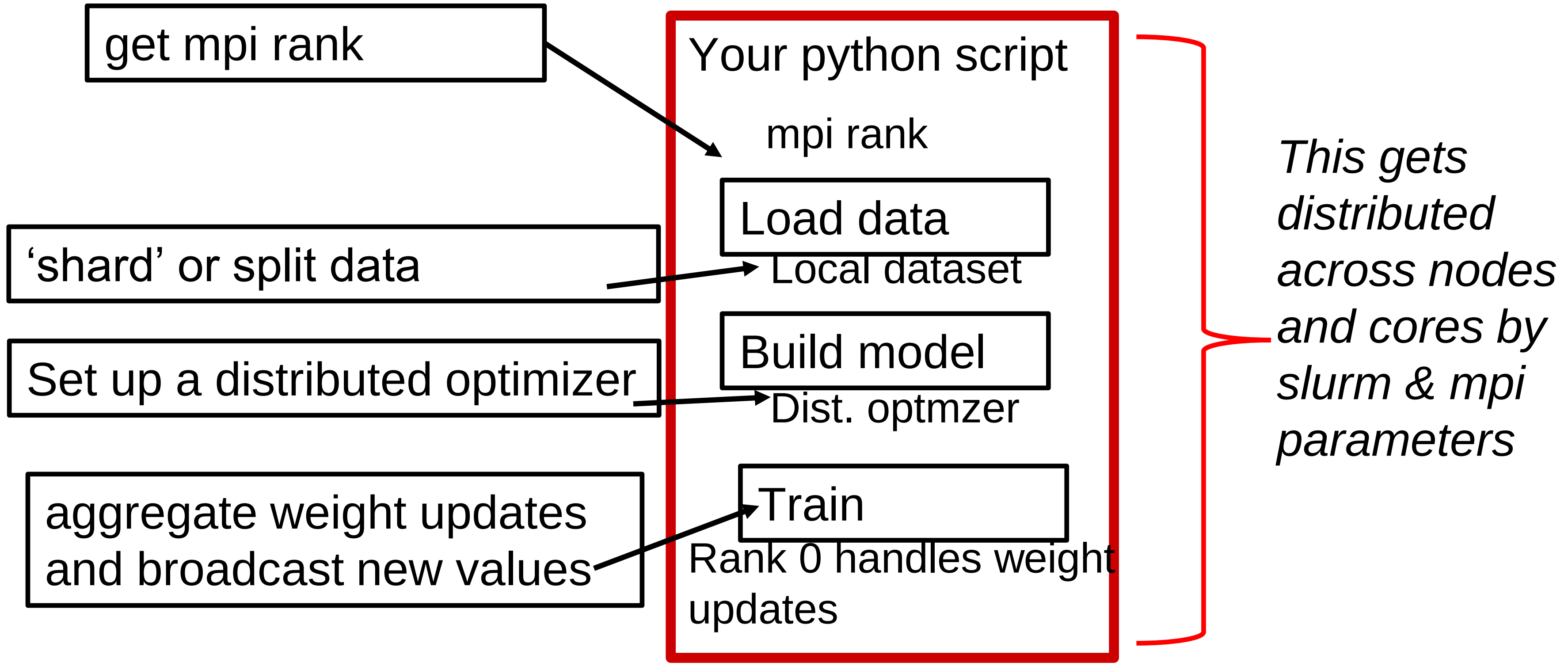
In slurm batch script:

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



Multinode, mpi launches instances

In slurm batch script:
mpirun –n **number of tasks** singularity → python


get mpi rank
'shard' or split data
Set up a distributed optimizer
aggregate weight updates and broadcast new values
Your python script
mpi rank
Load data
Local dataset
Build model
Dist. optmzer
Train
Rank 0 handles weight updates
This gets distributed across nodes and cores by slurm & mpi parameters

mpi launches one instance per processor

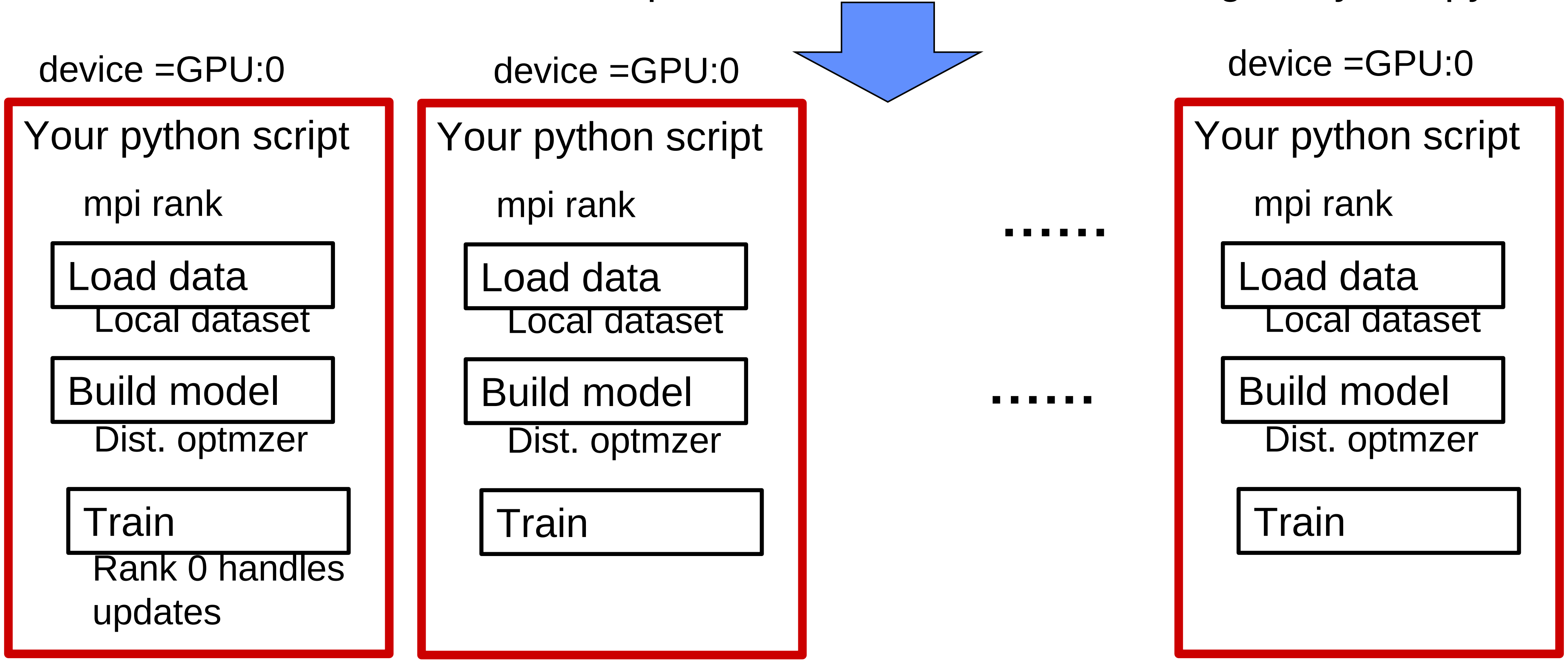
In slurm batch script:

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

mpi launches one instance per processor

In slurm batch script:

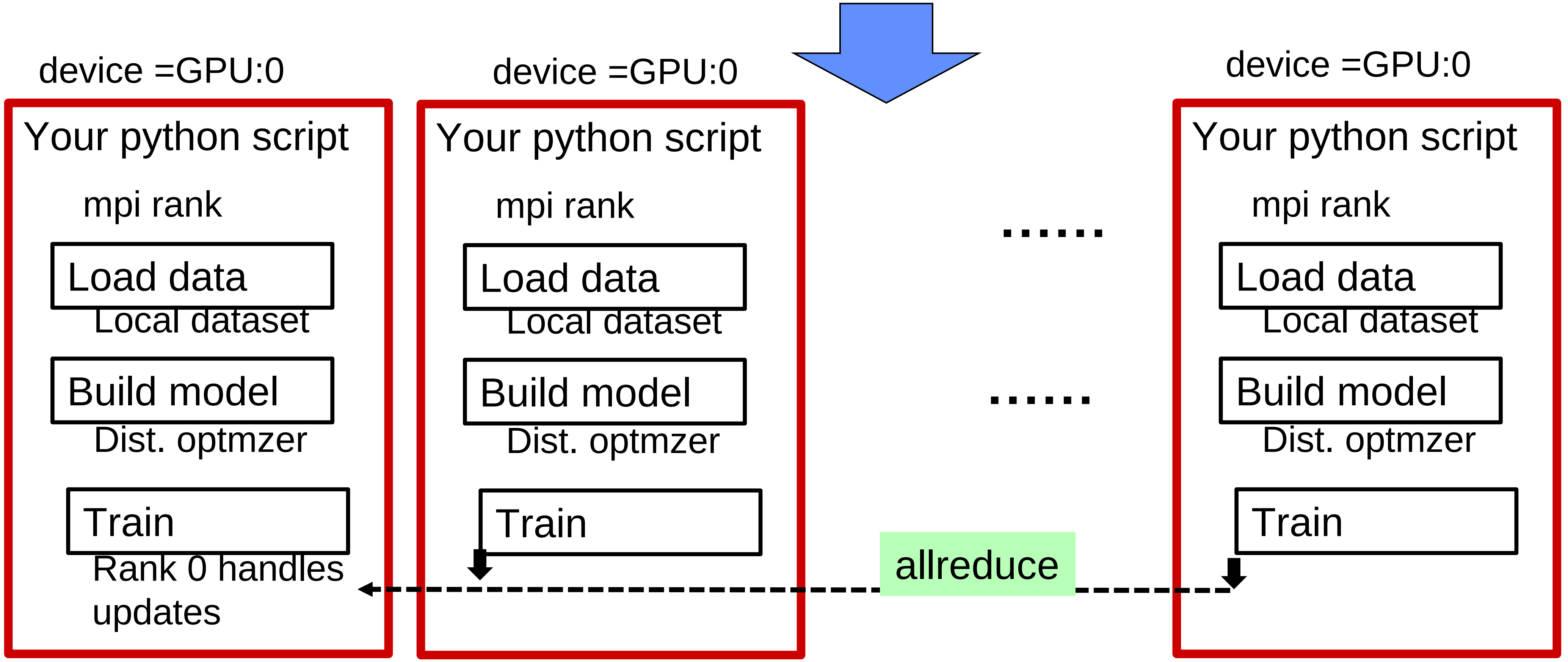
mpirun –n **number of tasks** singularity → python



For each batch: Horovod will aggregate & share weights updates

In slurm batch script:

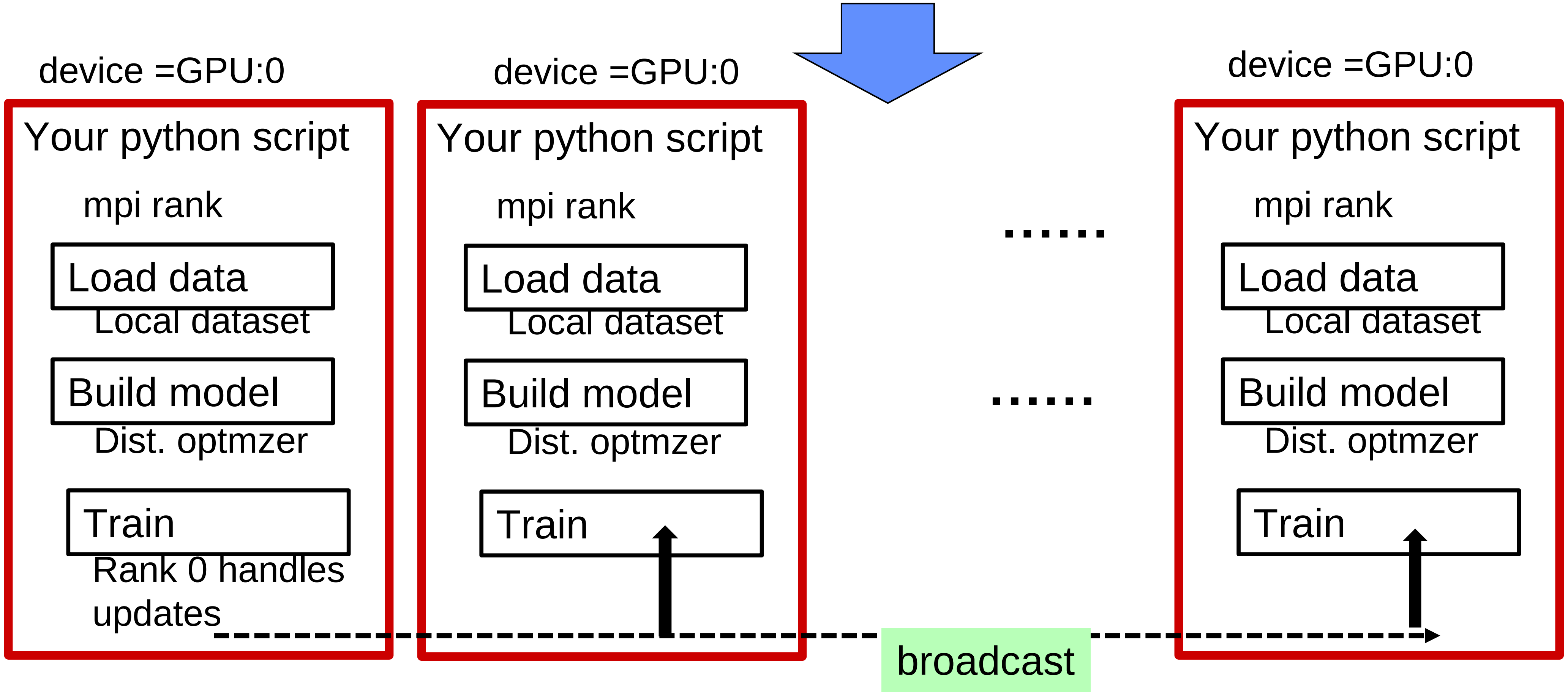
mpirun –n **number of tasks** singularity → python



For each batch: Horovod will aggregate & share weights updates

In slurm batch script:

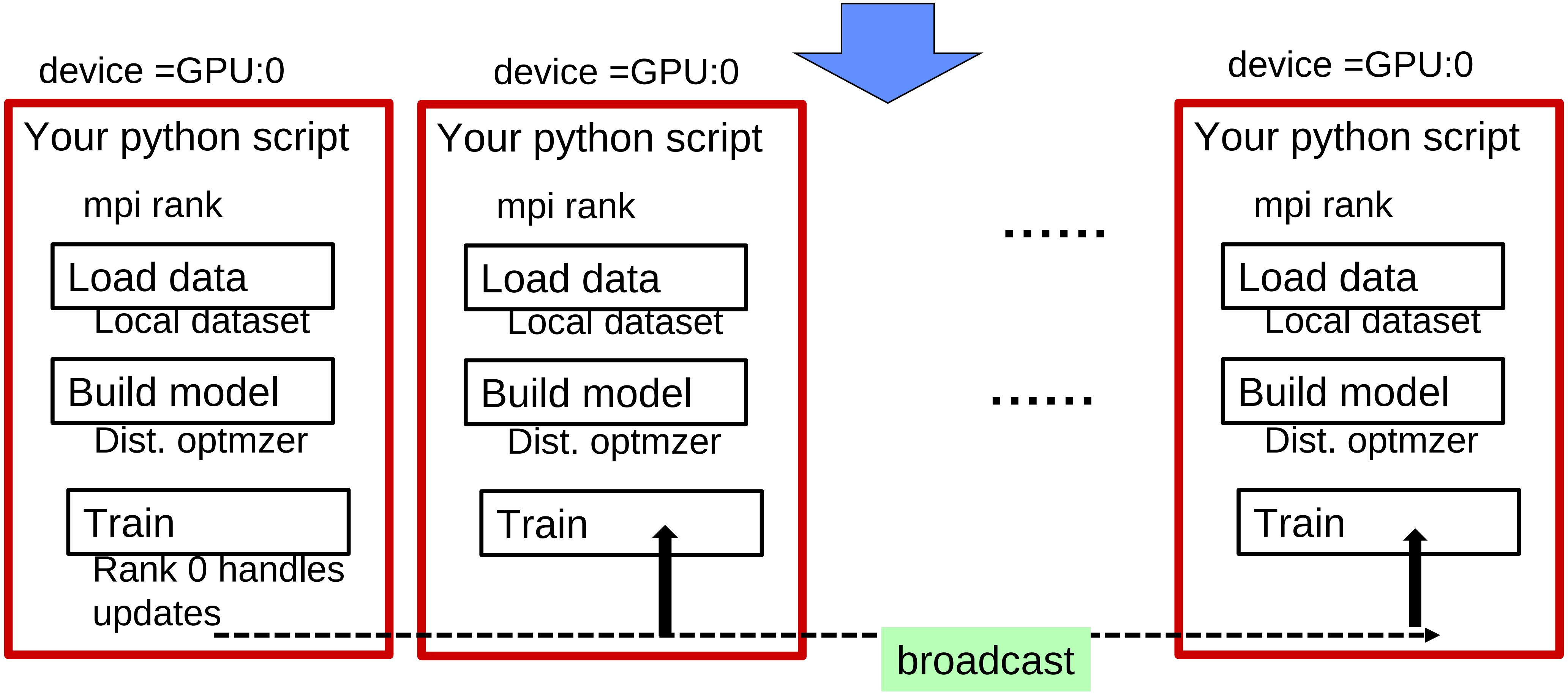
mpirun –n **number of tasks** singularity → python



For each 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

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

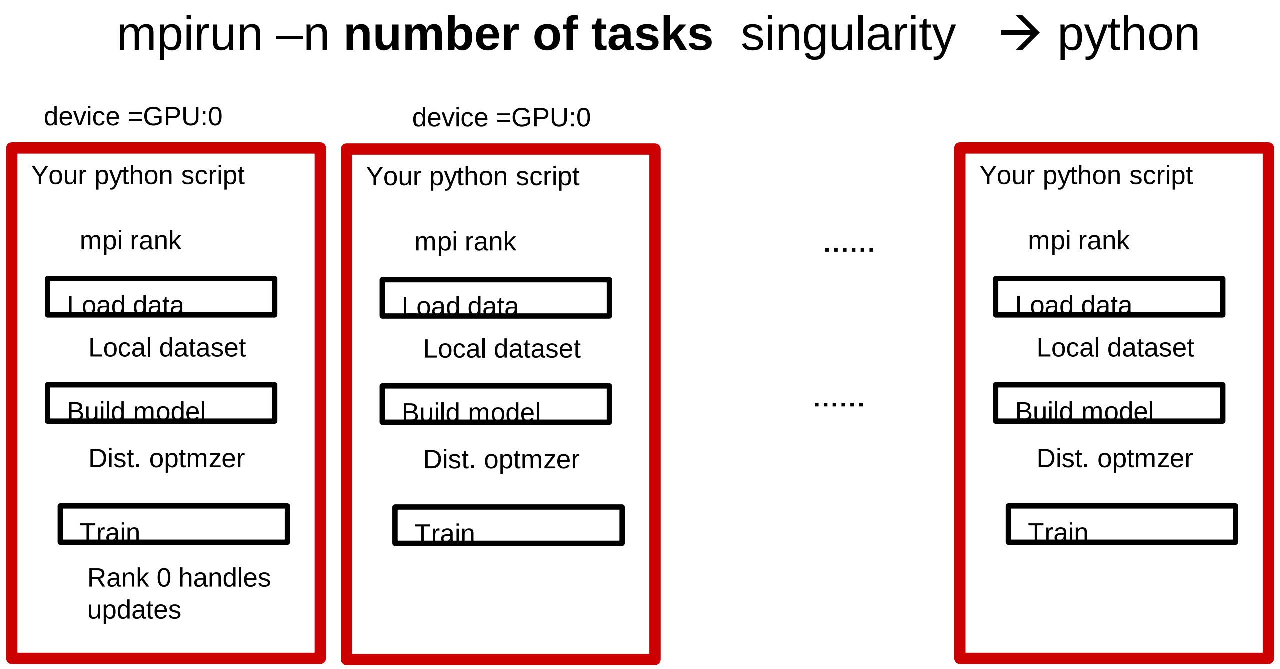
```
 optimizer2use  = hvd.DistributedOptimizer(……)
```

Horovod takes care of the communication for training and updating weights

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)



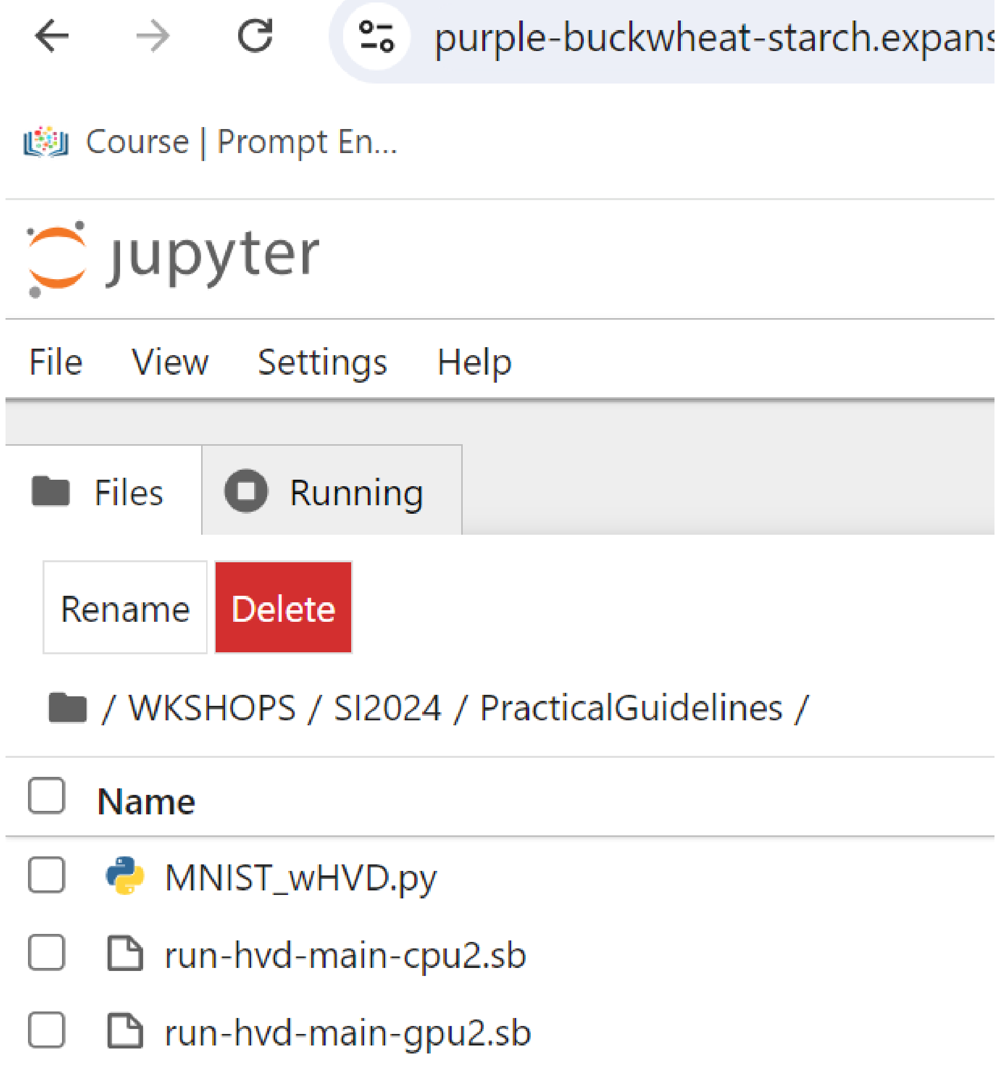
In terminal

```
$ jupyter-compute-tensorflow
```

In jupyter notebook session open the *MNIST_wHVD* notebook

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

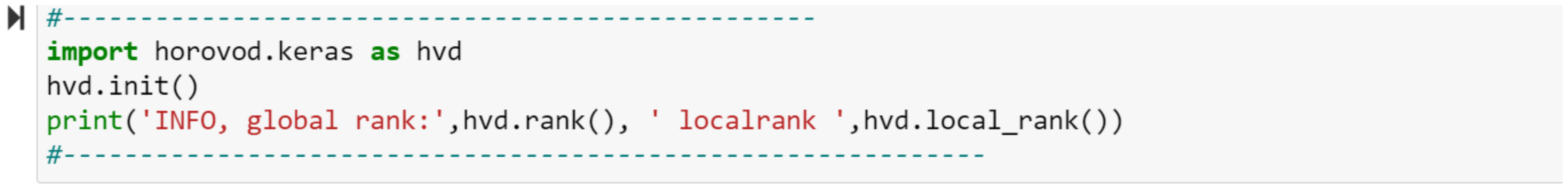
Follow instructions in the notebook and batch script



/ WKSHOPS / SI2024 / PracticalGuidelines /

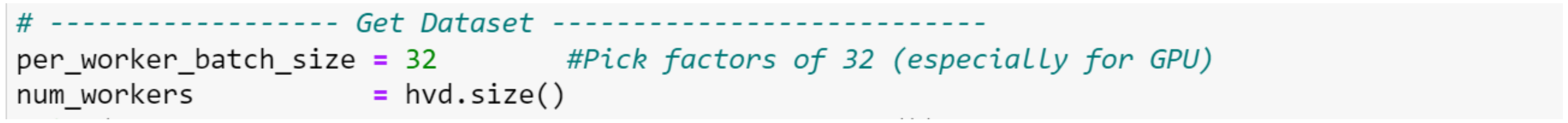
Name
MNIST_wHVD.py
run-hvd-main-cpu2.sb
run-hvd-main-gpu2.sb

Initialize back end communication, get mpi rank



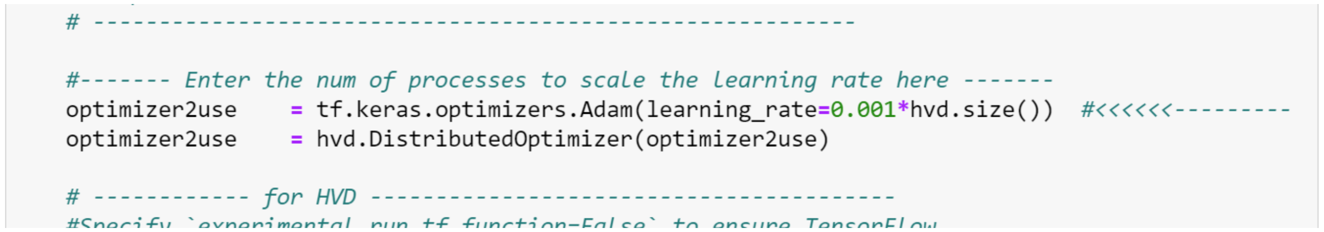
```
#-----------------------------------------------
import horovod.keras as hvd
hvd.init()
print('INFO, global rank:',hvd.rank(), ' localrank ',hvd.local_rank())
#-------------------------------------------------------
```

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

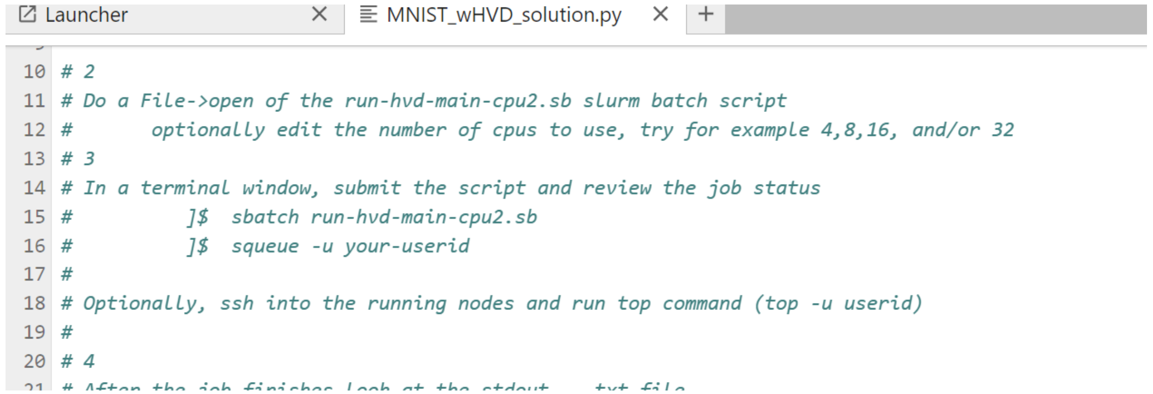


```
# ----------------------------------------------------------

#------- 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
```

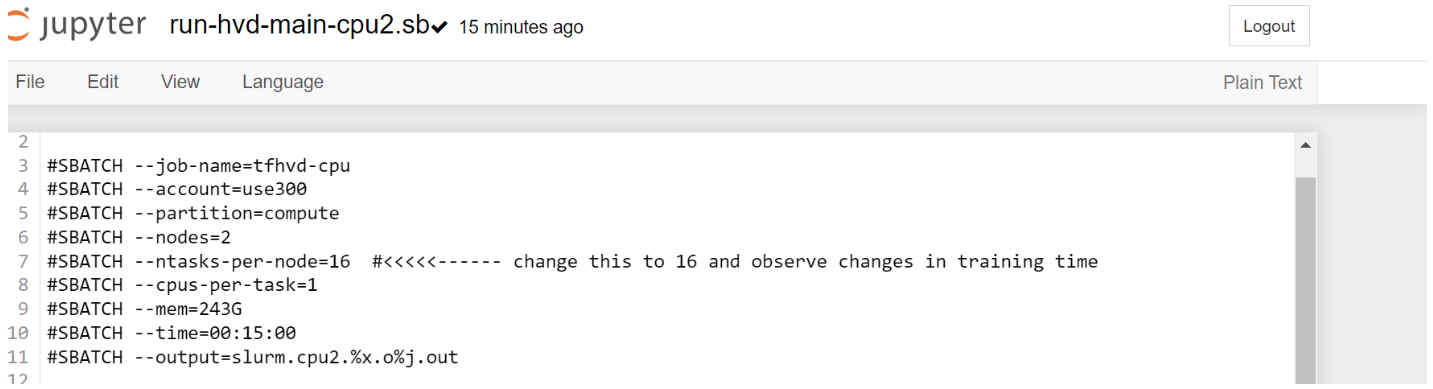
Your Task:

- run sbatch command for the slurm script (for cpu)

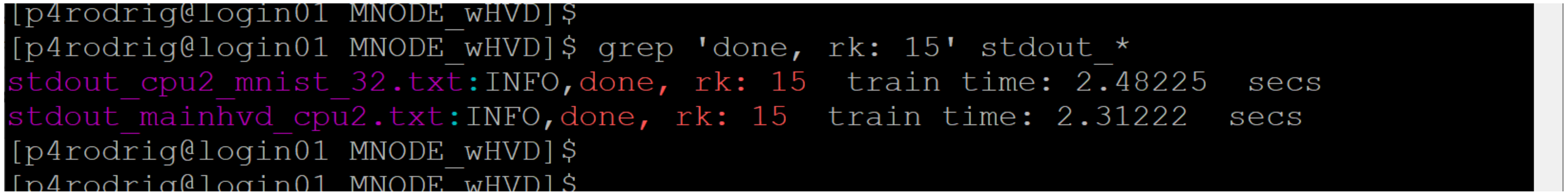
- change number of cpus to use, rerun, and review stdout output file



```
# 2
# Do a File->open of the run-hvd-main-cpu2.sb slurm batch script
#       optionally edit the number of cpus to use, try for example 4,8,16, and/or 32
# 3
# In a terminal window, submit the script and review the job status
#          ]$  sbatch run-hvd-main-cpu2.sb
#          ]$  squeue -u your-userid
#
# Optionally, ssh into the running nodes and run top command (top -u userid)
#
# 4
# After the job finishes look at the stdout   txt file
```

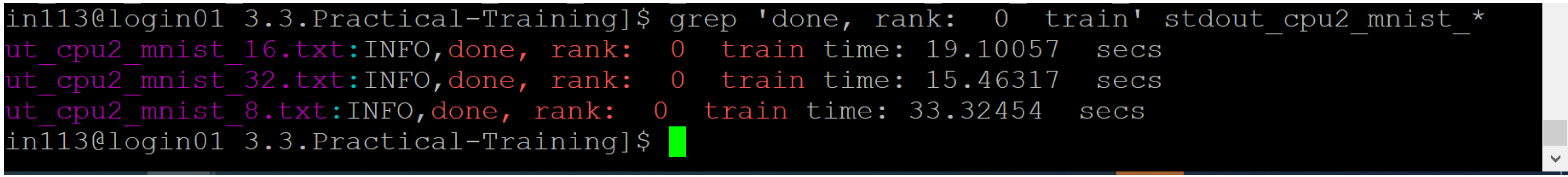


```

#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 '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]$
```

- **Pause   … you might find trade offs in communication vs cpus**



```
in113@login01 3.3.Practical-Training]$ grep 'done, rank:  0  train' stdout_cpu2_mnist_*
ut_cpu2_mnist_16.txt:INFO,done, rank:  0  train time: 19.10057  secs
ut_cpu2_mnist_32.txt:INFO,done, rank:  0  train time: 15.46317  secs
ut_cpu2_mnist_8.txt:INFO,done, rank:  0  train time: 33.32454  secs
in113@login01 3.3.Practical-Training]$
```

- **End**