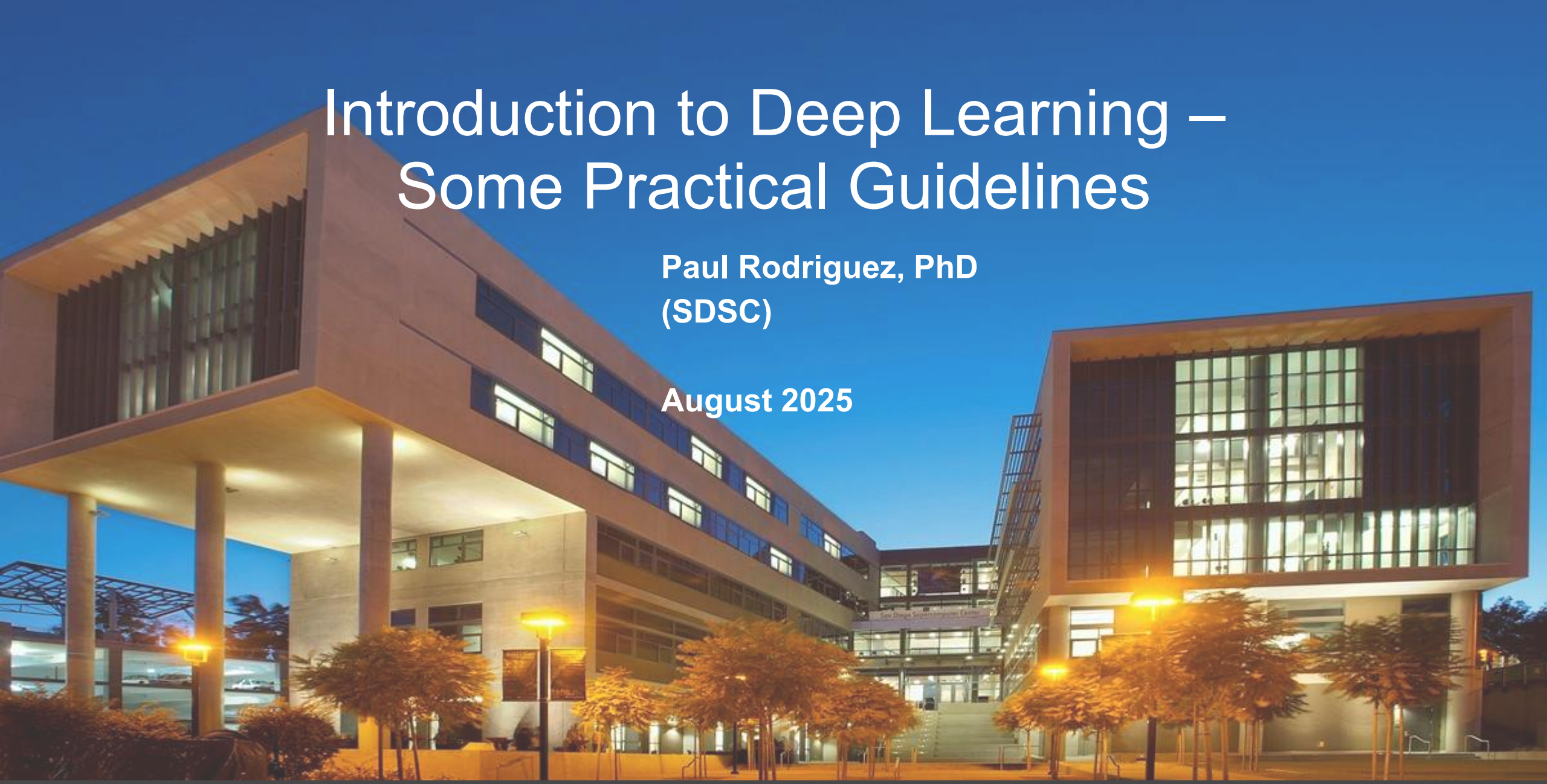
Introduction to Deep Learning – Some Practical Guidelines

Paul Rodriguez, PhD
(SDSC)

August 2025



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/Multidevice Execution

Exercise/Demo, Multinode MNIST

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

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

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

Hyperparameter Search Tool

- Several search strategies, such as:

Hyperband is like a tournament of hyperparameter configurations

- Typically, you usually wrap the model build function with another function, and add in arguments for the hyperparameters

Workflow and Organizing Jobs

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

Model Level: run & test model for each parameter configuration

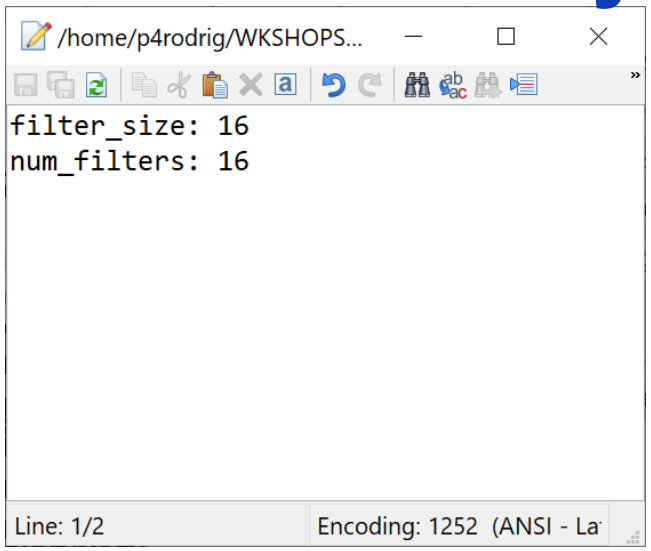
Data Level: loop through cross validation datasets (if applicable)

- **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



Read file as python dictionary

```yaml
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)
```

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**

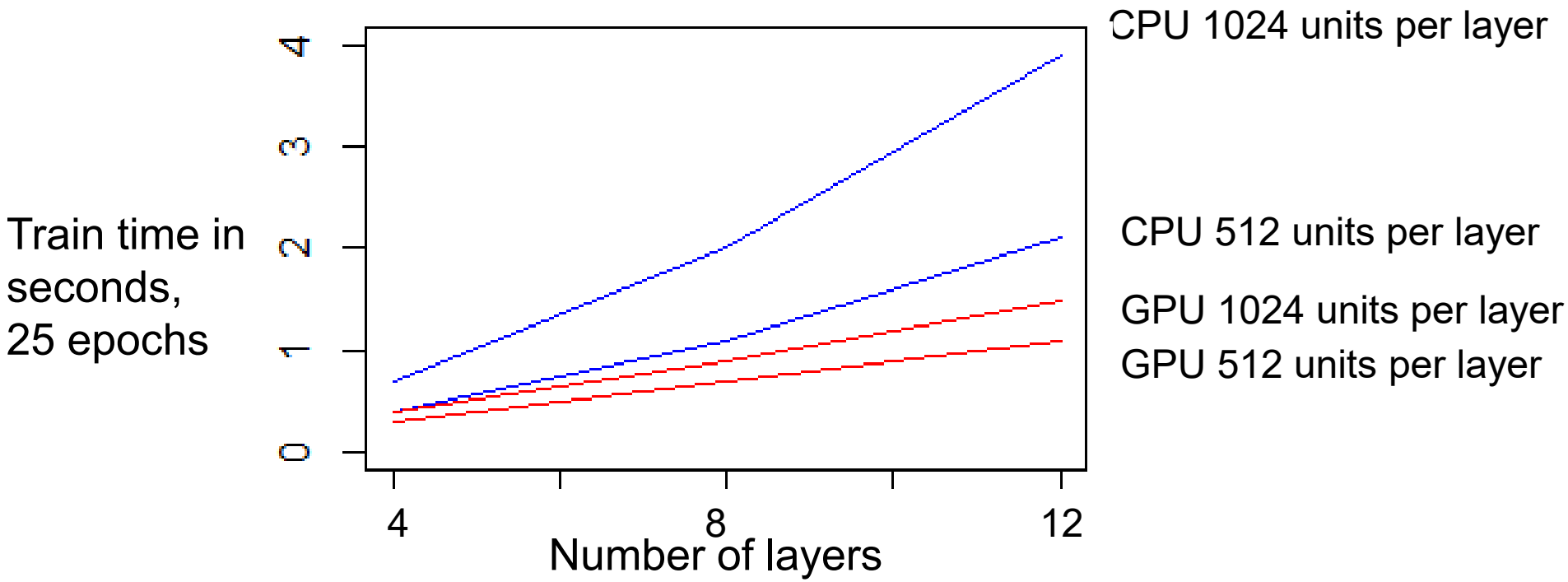
Code snippet to check for GPU devices

/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

Scaling in Deep Learning

- Two Goals

1 Speed Up Learning - parallelizing data and/or model

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 so that layers are on different devices, ie inter-layer partitions (you organize layers)

- Tensor Parallelism: intra-layer partitions (model has to support it)

Parallel DL models with multiple nodes/devices

- **Data Parallel:**

1. Launch your script on each device on each node

2. Split up data

3. Each device trains a copy of the model with a part of the data

4. Aggregate parameter updates across devices/nodes

Parallel DL models with multiple nodes/devices

- **Data Parallel:**

1. Launch your script on each device on each node
2. Split up data
3. Each device trains a copy of the model with a part of the data
4. Aggregate parameter updates across devices/nodes

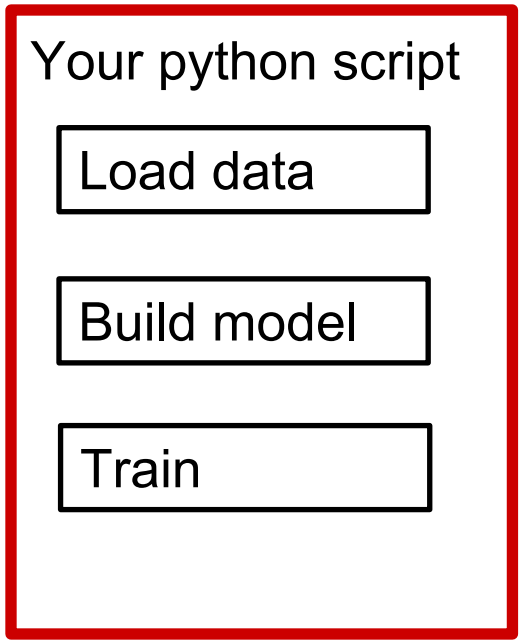
- **Main functions:**

1. *mpirun* command (also *torchrun* exists but I haven't tested it much)
2. *Pytorch data loader will parallelize sampling*
3. *Pytorch DDP (**Distributed Data Parallel**) will wrap a model*
4. *DDP wil use the Pytorch 'init_backend' function*

For example, single node, single device execution

In slurm batch script:

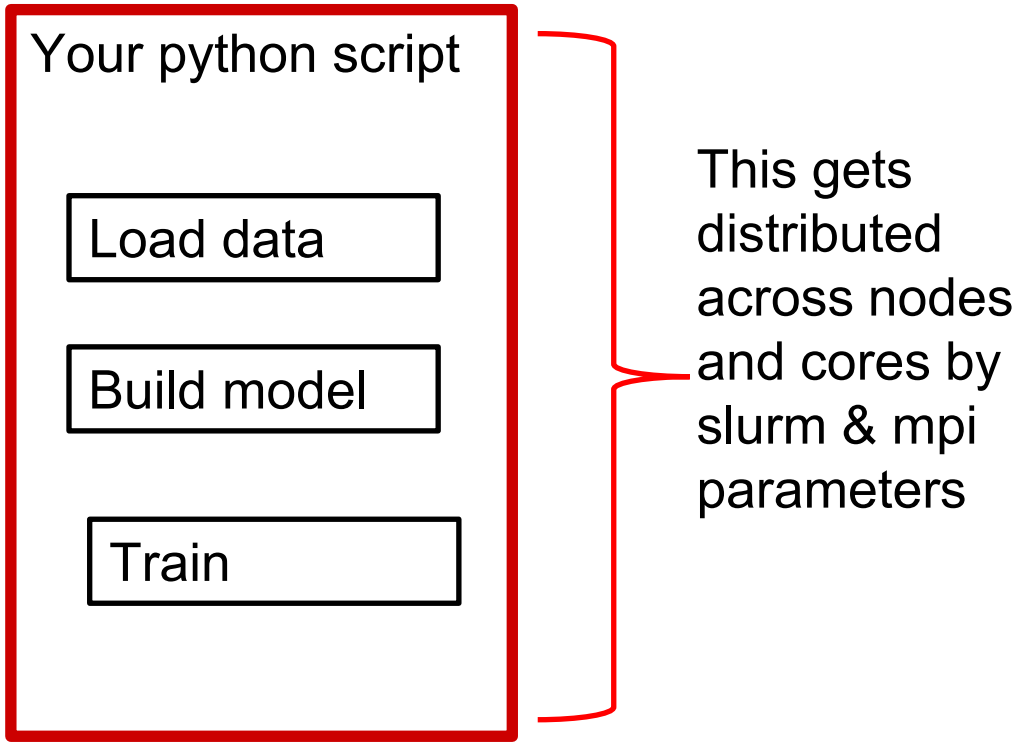
singularity → python



Multinode, mpi launches instances

In slurm batch script:

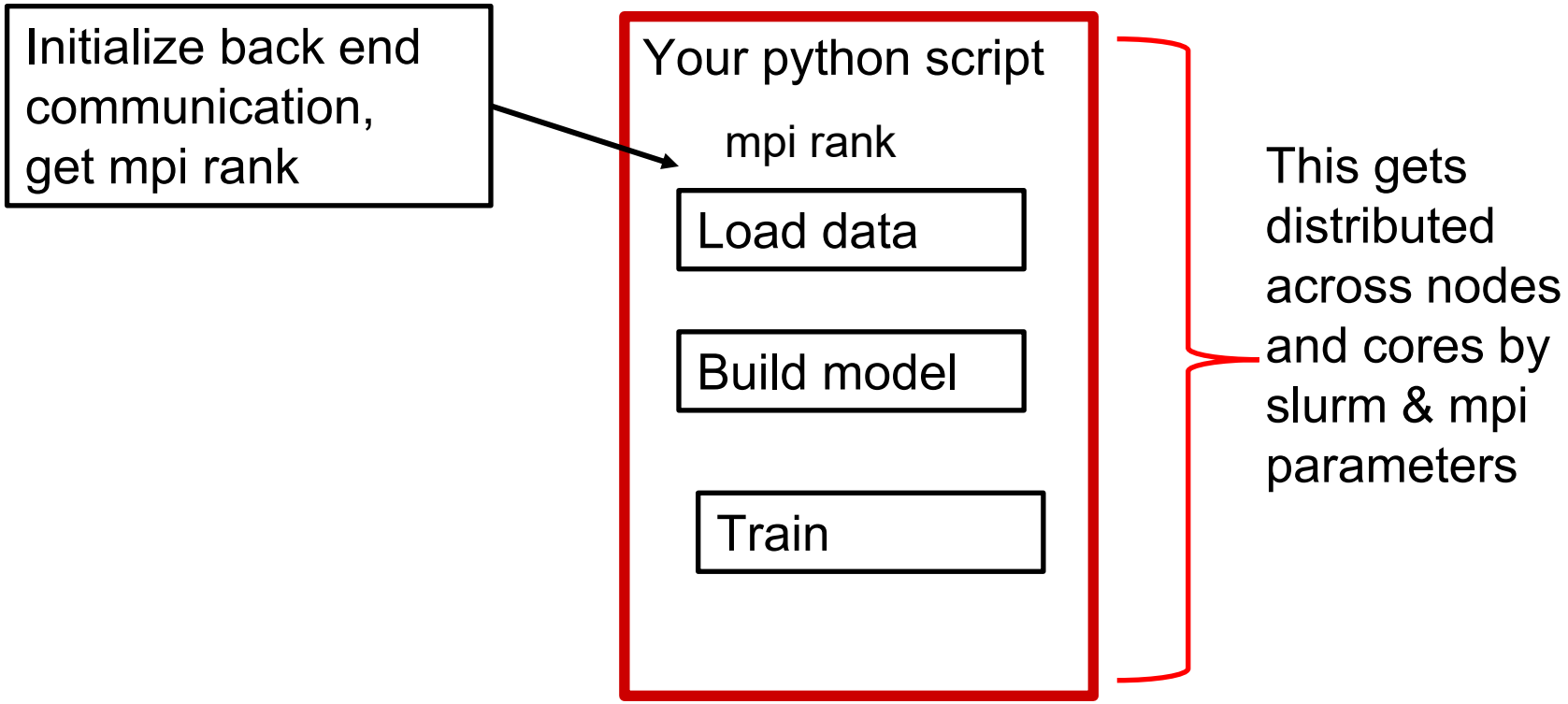
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

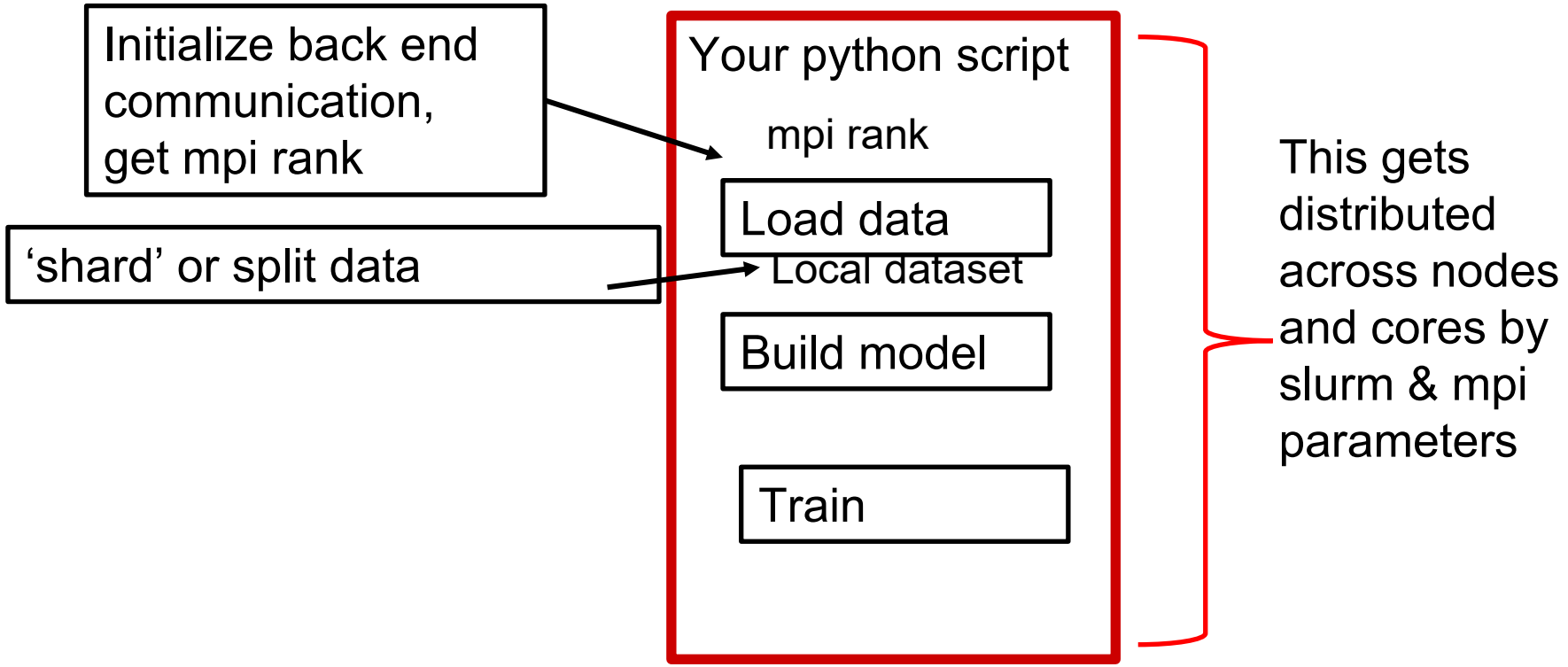
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

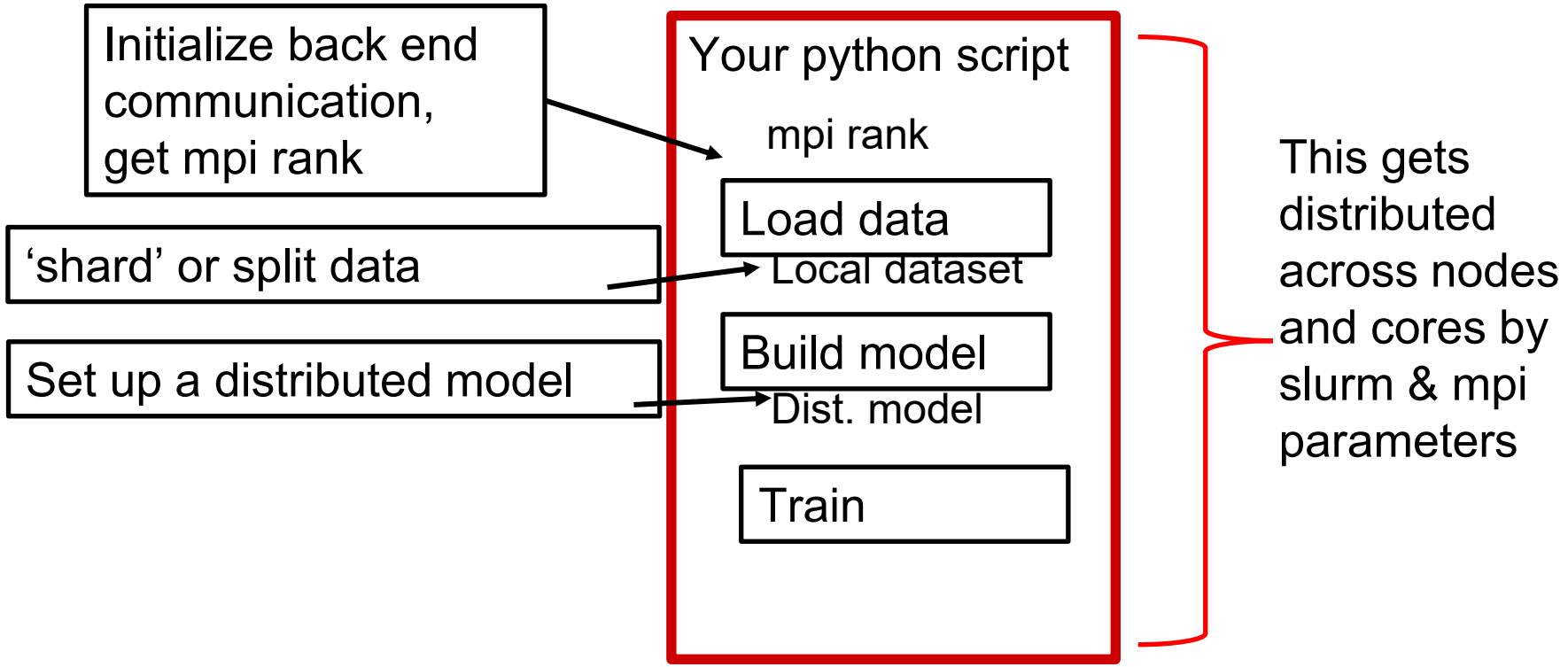
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

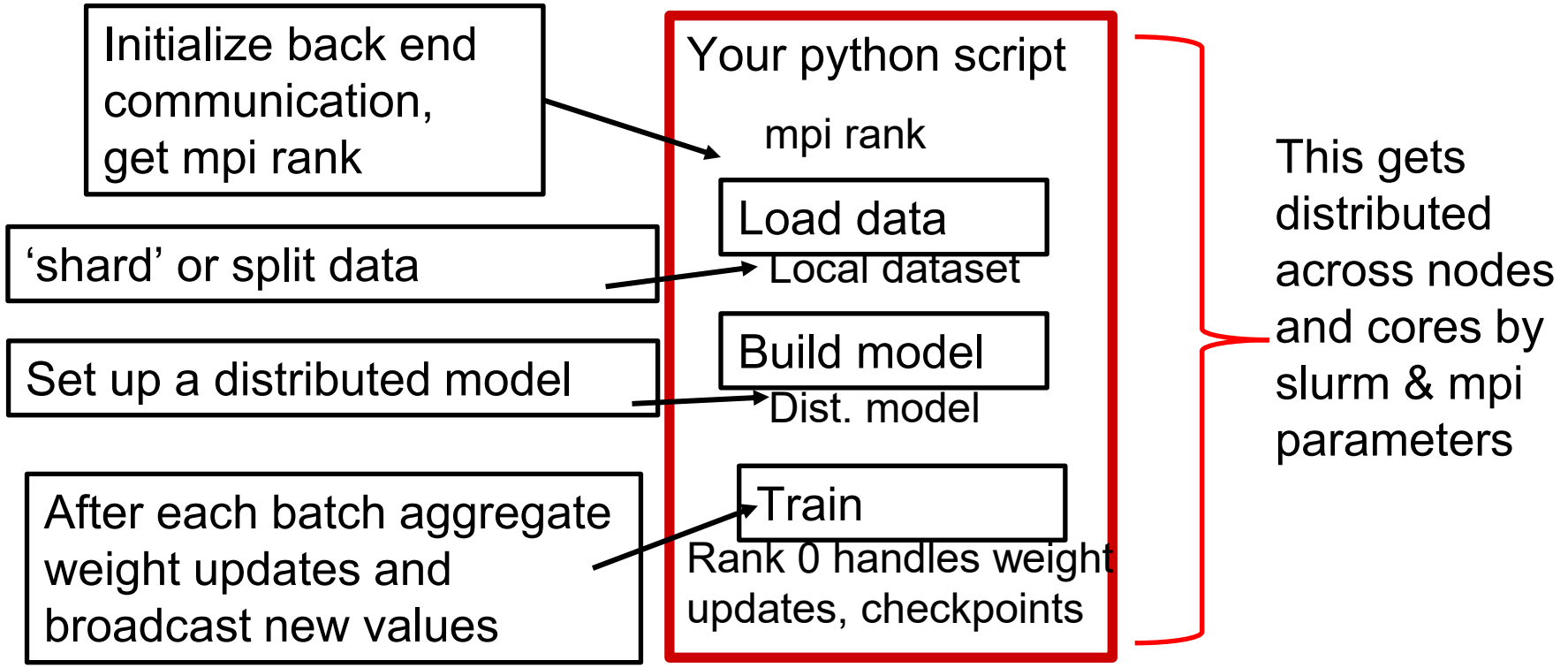
mpirun –n **number of tasks** singularity → python



Multinode, mpi launches instances

In slurm batch script:

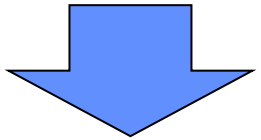
mpirun –n **number of tasks** singularity → python



mpi launches one instance per processor

In slurm batch script:

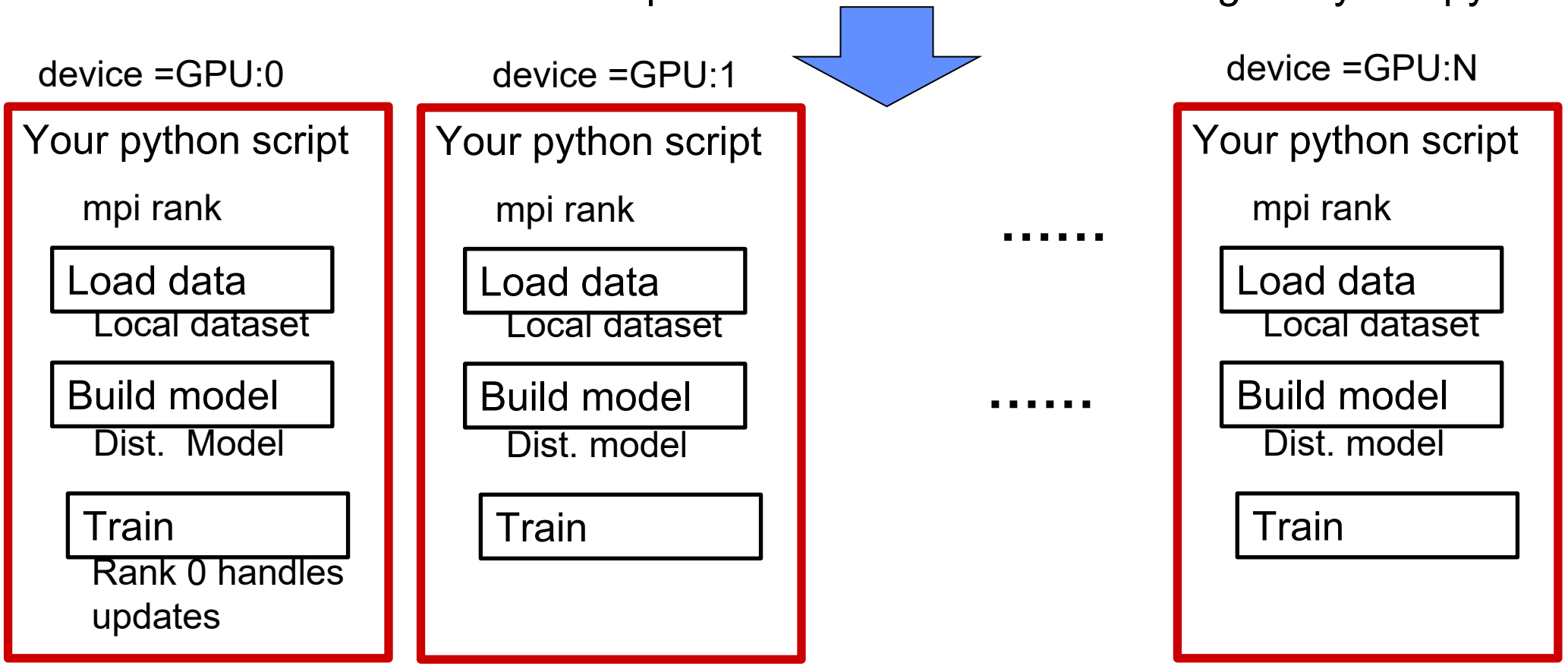
mpirun –n **number of tasks** singularity → python



mpi launches one instance per processor

In slurm batch script:

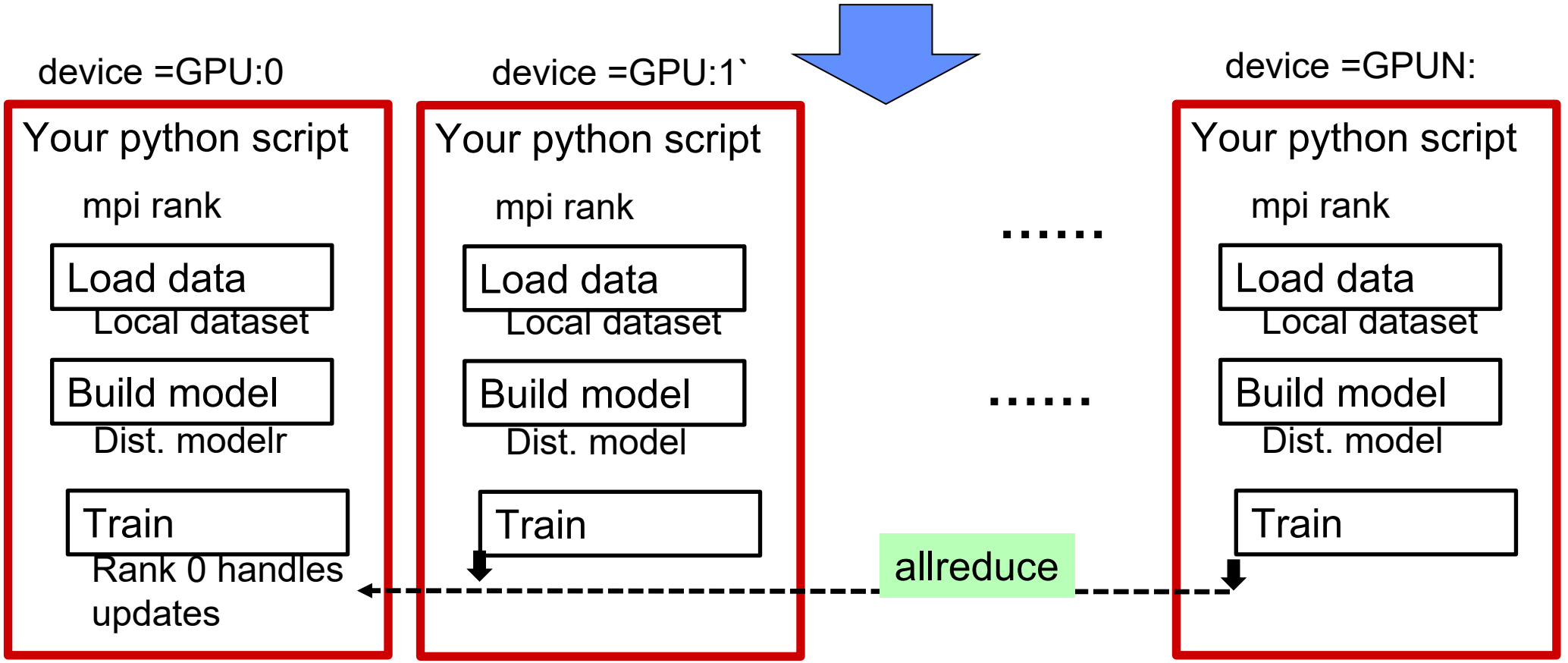
mpirun –n **number of tasks** singularity → python



For each batch: DDP will aggregate & share weights updates

In slurm batch script:

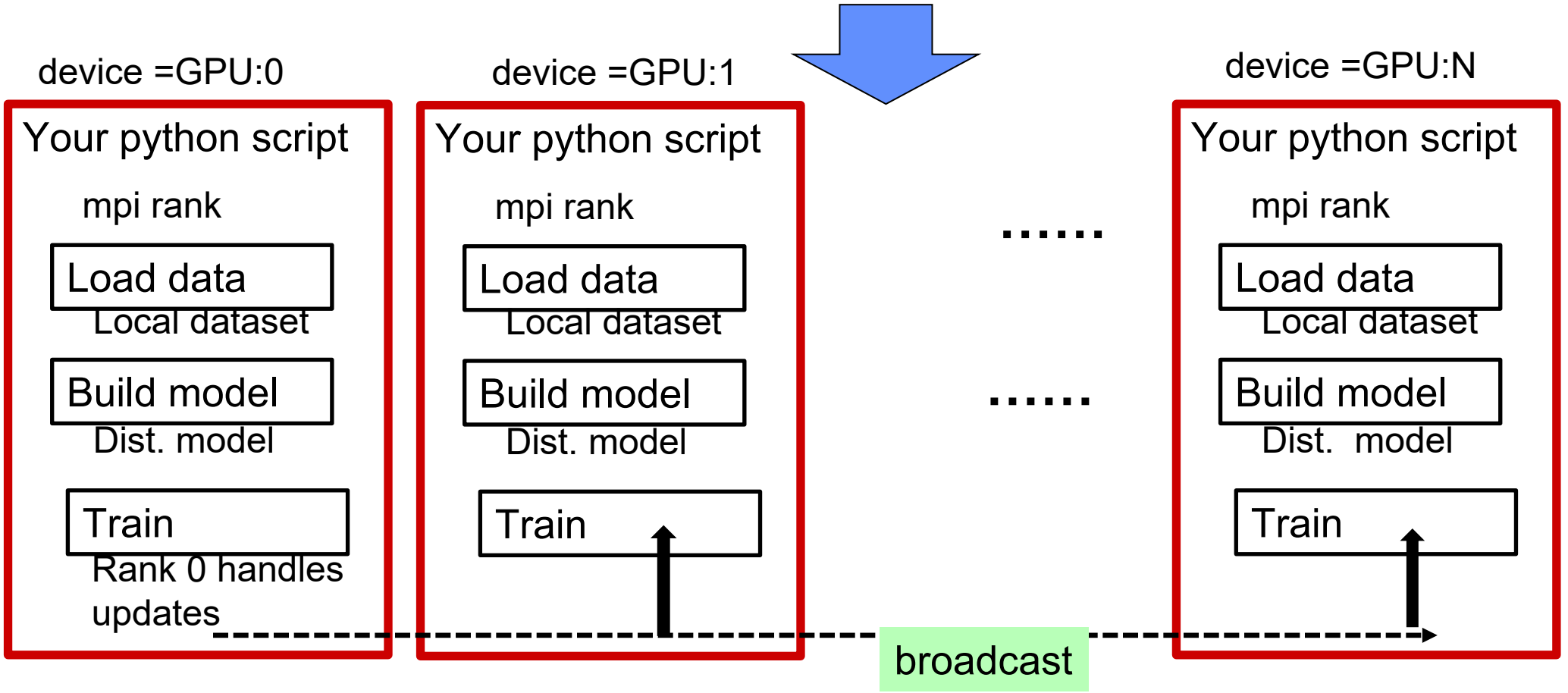
mpirun –n **number of tasks** singularity → python



For each batch: DDP will aggregate & share weights updates

In slurm batch script:

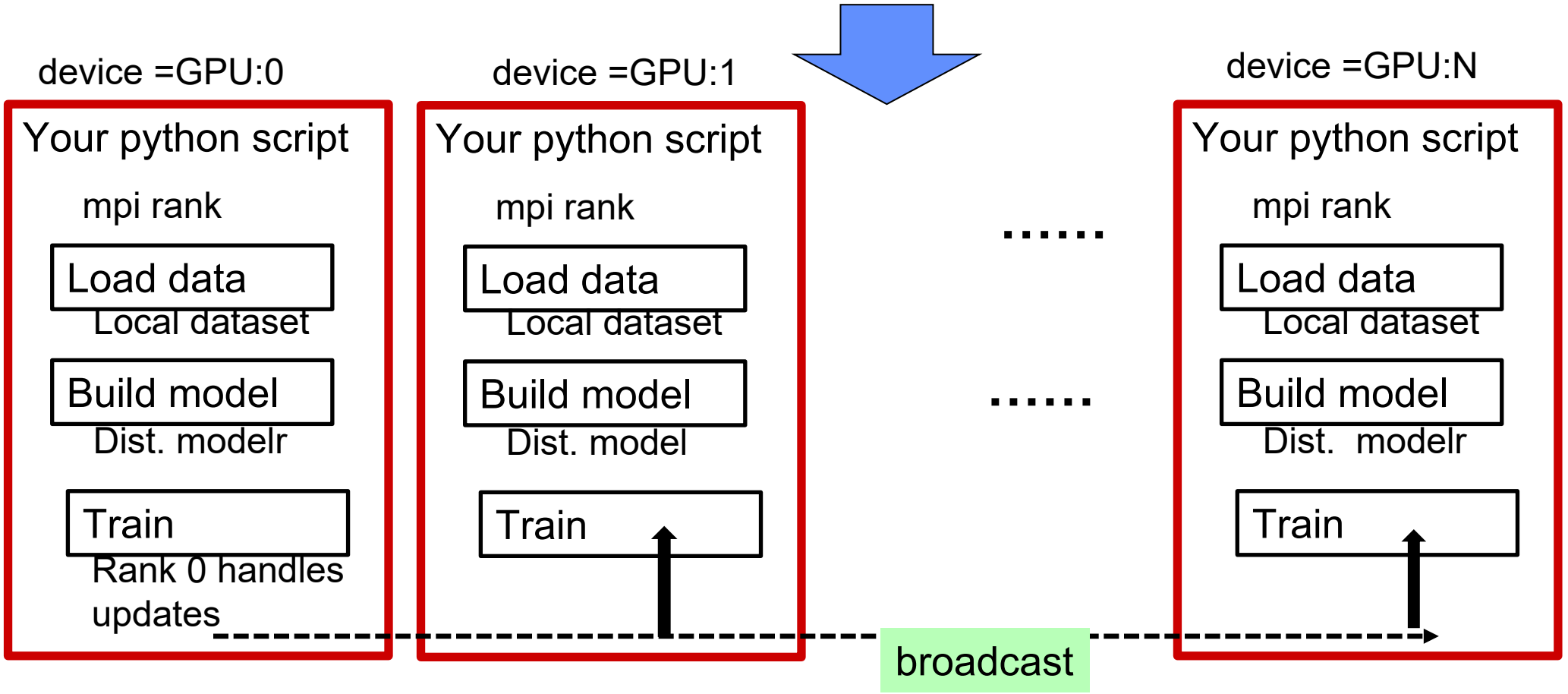
mpirun –n **number of tasks** singularity → python



For each batch: DDP 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 – Pytorch functions

Initialize back end communication, get mpi rank

```
world_size     = int(os.environ['OMPI_COMM_WORLD_SIZE'])
rank           = int(os.environ['OMPI_COMM_WORLD_RANK'])
local_rank     = int(os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
device         = torch.cuda.set_device(local_rank)
torch.distributed.init_process_group('nccl',rank=rank,world_size=world_size
```

'shard' or split data

```
train_sampler = torch.utils.data.distributed.DistributedSampler(dataset1)
train_loader    =torch.utils.data.DataLoader(dataset1,
                        batch_size =  batch_size,
                        sampler      = train_sampler, …)
```

Set up a distributed model

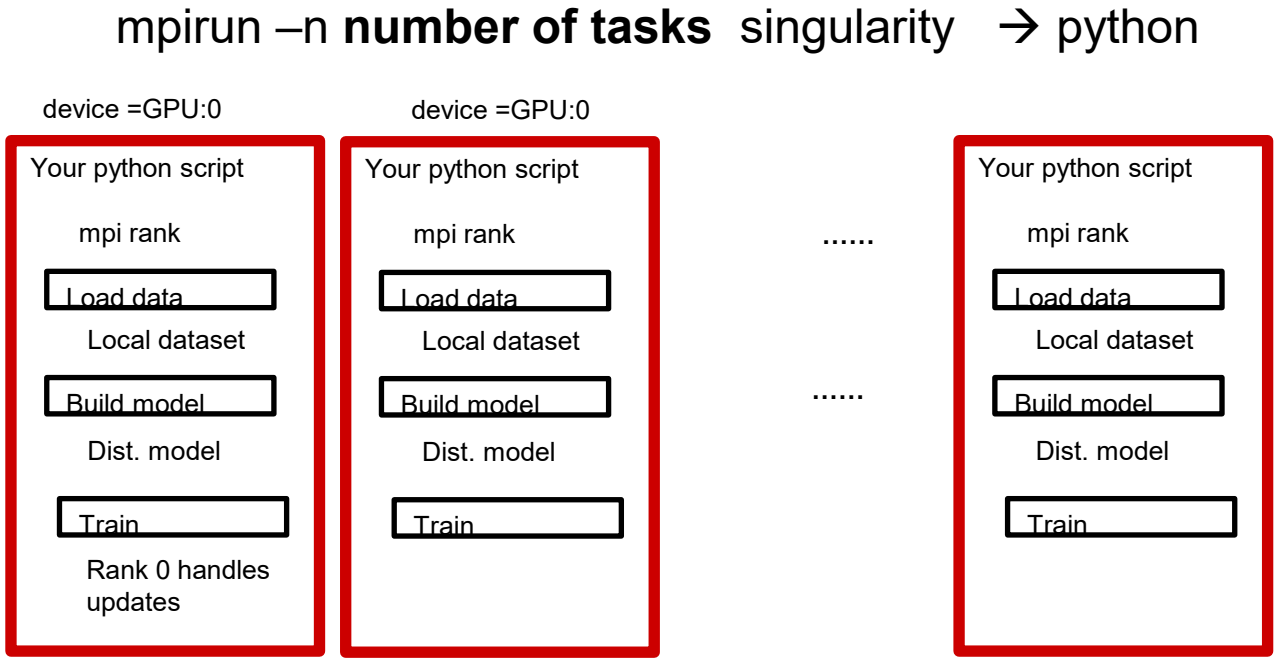
```
model = Net().to(device)
model = torch.nn.parallel.DistributedDataParallel()   #with default args
```

After each batch aggregate weight updates and broadcast new values

Exercise, multinode MNIST execution

- **Goal: Get familiar with Pytorch coding for multinode execution on Expanse**
- **Goal: Get familiar with slurm batch script multinode parameters**
- **Let's login to a terminal window**

(see next page for tasks to try)



SLURM batch script highlights

- **Set up job resources**

- **Set up 'master' ip address**

- **Use mpirun command**

  2*8=16

```
#!/usr/bin/env bash
#SBATCH --job-name=pyt-cpu
#SBATCH --account=gue998
#SBATCH --partition=compute
#SBATCH --nodes=2                #try 1 or 2
#SBATCH --ntasks-per-node=8   #try 8 or 16 or 32 etc..
....

#set up ip addresses for communication
declare -xr MASTER_ADDR=$(mpirun --allow-run-as-root -n 1 hostname -i
declare -xr MASTER_PORT=${MASTER_PORT:-15566};

#use -n num-of-nodes * num-per-node
mpirun -n 16 -npernode 8  singularity exec --bind /expanse,/scratch
/cm/shared/apps/containers/singularity/pytorch/pytorch-latest.sif pyth
stdout_mnist_multinode_cpu.txt
```

Your task:

- run sbatch command for the slurm script

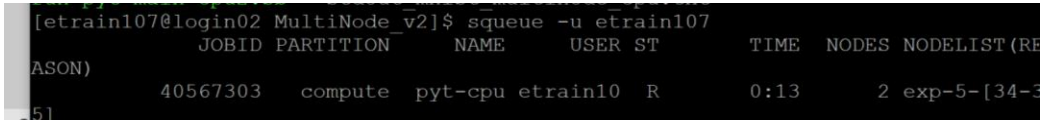
$ sbatch run-pyt-main-cpu2.sb

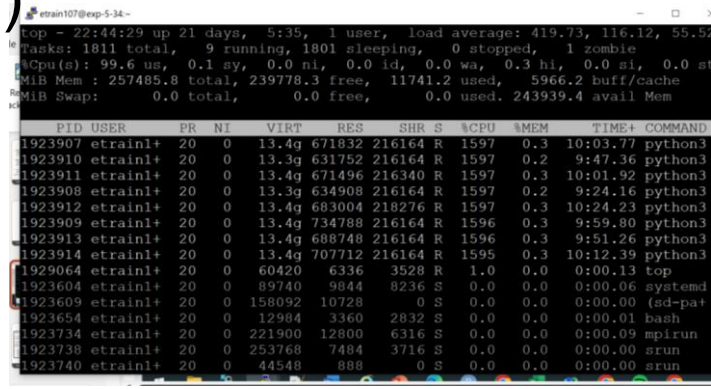
- ssh into node to see execution

$ squeue –u $USER (this will show your nodes)

$ ssh exp-##-## (this will connect to a node)

[exp-##-##] $ top –u $USER (this will show what's running)

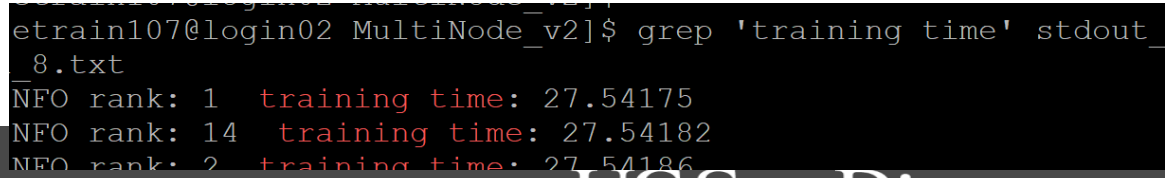




- review stdout output file

$ grep 'training time' stdout_mnist_...txt

- homework: find trade offs in speed up vs communication



- **pause**