

Deep Learning on Supercomputers

An introduction

Fahad Khalid [f.khalid@fz-juelich.de]. Simulation and Data Laboratory Neuroscience, Jülich Supercomputing Centre.

November 25, 2021. Course: Introduction to Supercomputing at JSC - Theory & Practice.

Agenda

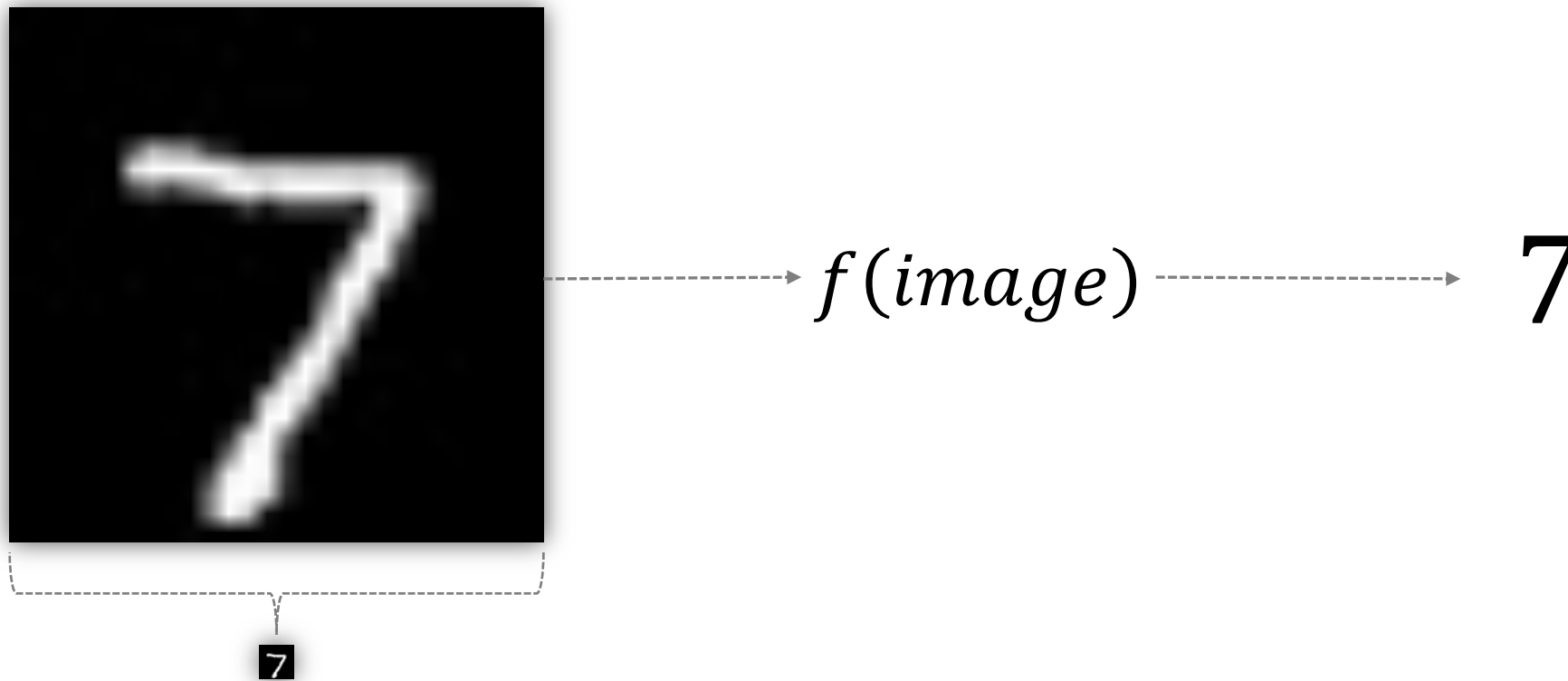
1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

Agenda

1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

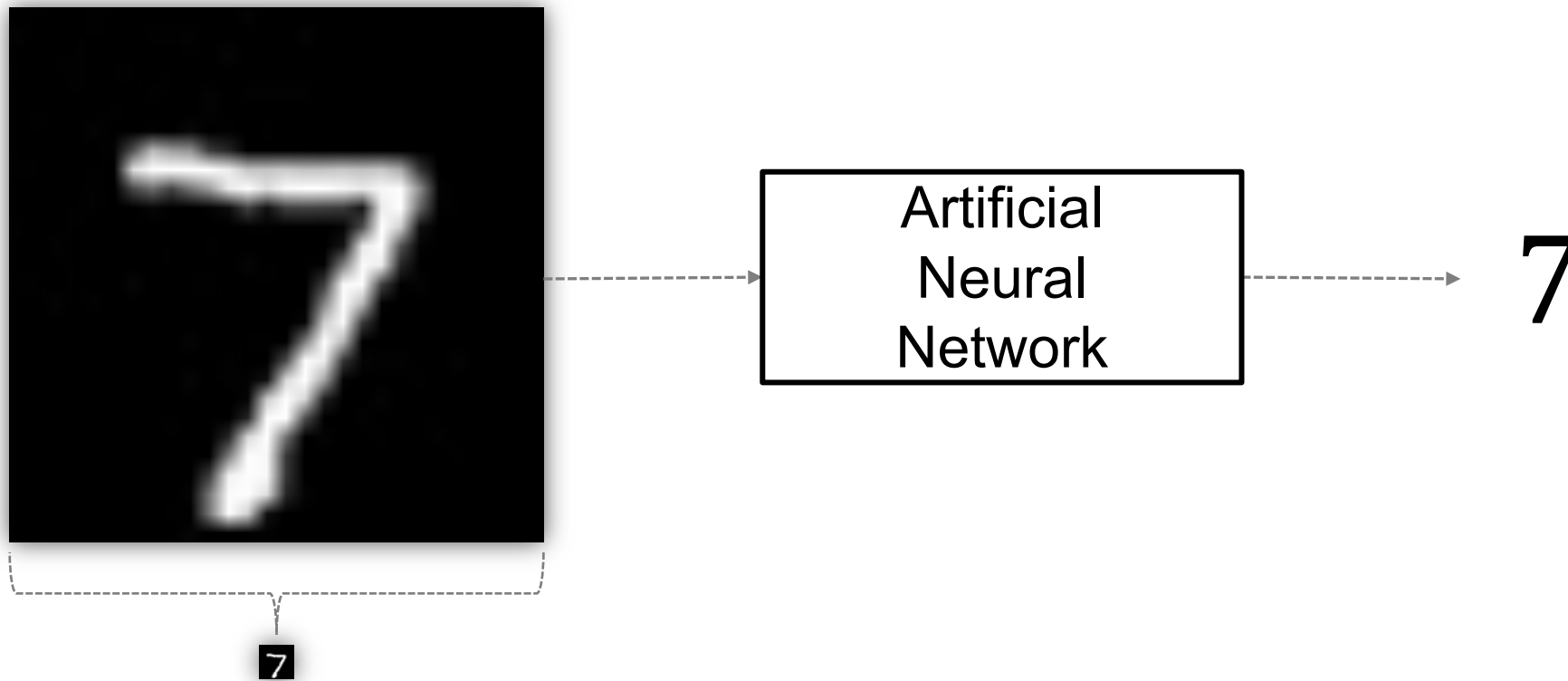
Example

Handwritten digit recognition



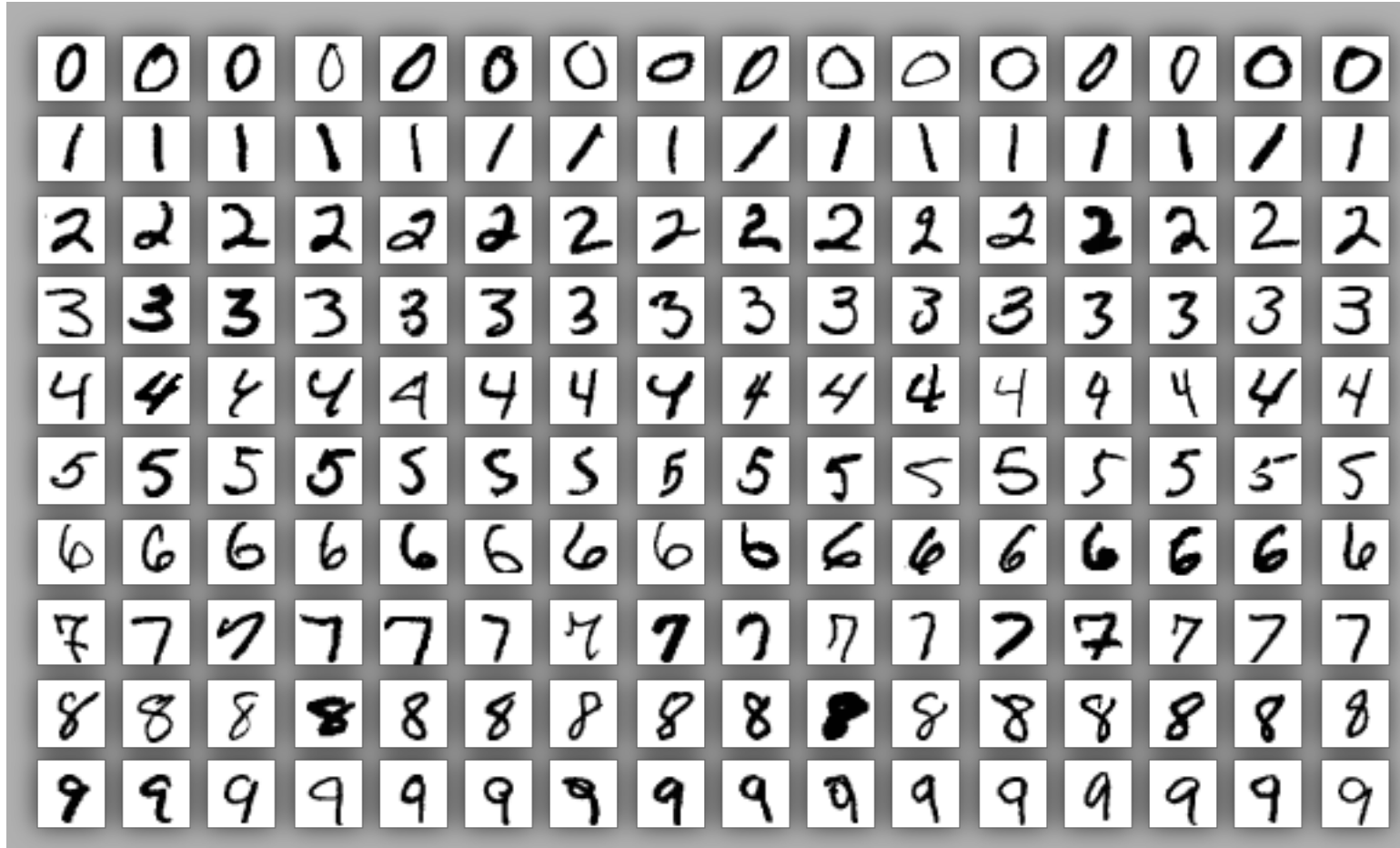
Example

Handwritten digit recognition



Example

Modified National Institute of Standards and Technology (MNIST) database

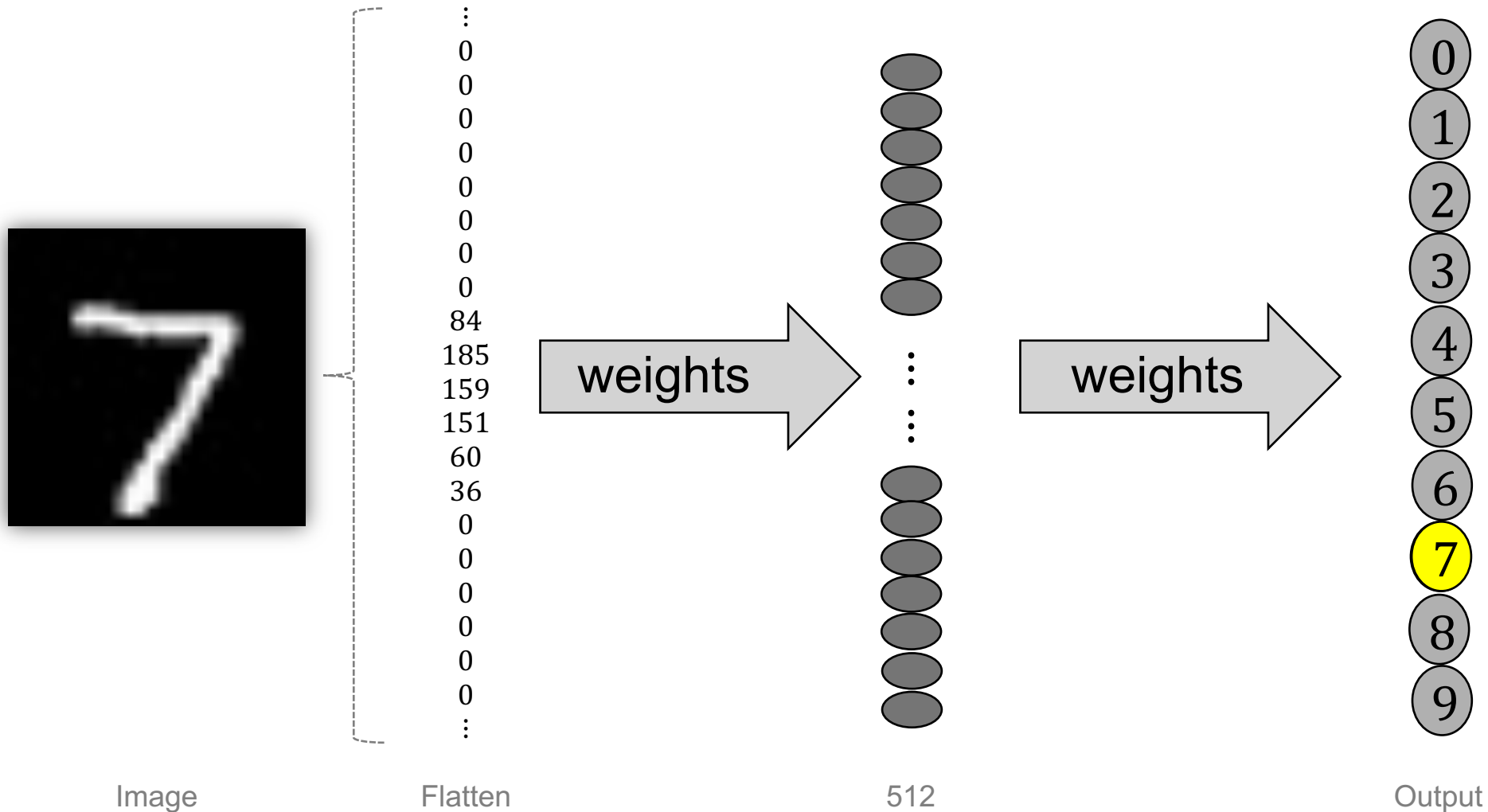


- Image dimensions: 28×28
- Each pixel $p \in [0, 255]$
- 60,000 training examples
- 10,000 test examples

Source: Modified version of [this](#). [License](#).

Example

A basic network for classification



Example

MNIST classification with tf.keras: Code

```
1. import tensorflow as tf
2. mnist = tf.keras.datasets.mnist
3. (x_train, y_train), (x_test, y_test) = mnist.load_data()
4. x_train, x_test = x_train / 255.0, x_test / 255.0
5. model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
6. optimizer = tf.keras.optimizers.Adam()
7. epochs = 4
```

```
8. model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
9. model.fit(
    x=x_train,
    y=y_train,
    batch_size=32,
    epochs=epochs
)
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

Example

MNIST classification with tf.keras: Output

```
$ python -u mnist_single_gpu.py
```

```
Epoch 1/4
```

```
1875/1875 [=====] - 8s 131us/sample - loss: 0.2006 - acc: 0.9404
```

```
Epoch 2/4
```

```
1875/1875 [=====] - 7s 120us/sample - loss: 0.0815 - acc: 0.9749
```

```
Epoch 3/4
```

```
1875/1875 [=====] - 7s 120us/sample - loss: 0.0548 - acc: 0.9831
```

```
Epoch 4/4
```

```
1875/1875 [=====] - 7s 119us/sample - loss: 0.0376 - acc: 0.9879
```

```
Test loss: 0.06436453427168308
```

```
Test accuracy: 0.9805
```

```
1. score = model.evaluate(x=x_test, y=y_test, verbose=0)
2. print(f'Test loss: {score[0]}')
3. print(f'Test accuracy: {score[1]}')
```

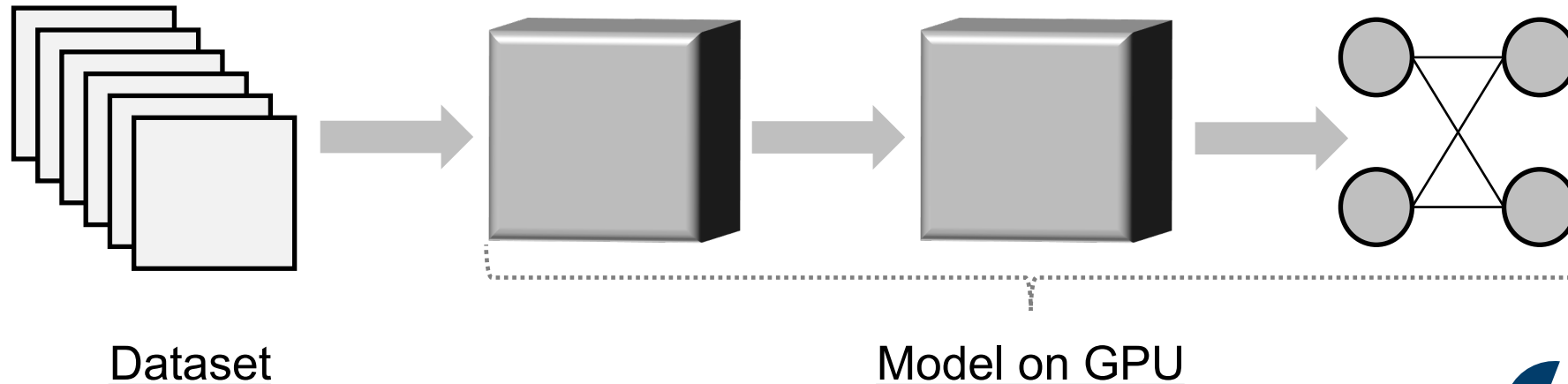
Agenda

1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

Distributed training

Motivation

1. Train faster
 - i. Large dataset size
 - ii. Compute intensive model
2. Use a dataset with very large instances
3. Use a very large model
4. Increase the effective batch size



Distributed training

Effective batch size

Specified batch size: 4

GPU 0

Distributed training

Effective batch size

Specified batch size: 4

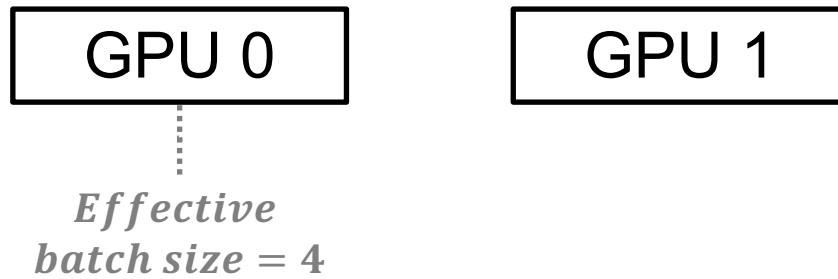
GPU 0

*Effective
batch size = 4*

Distributed training

Effective batch size

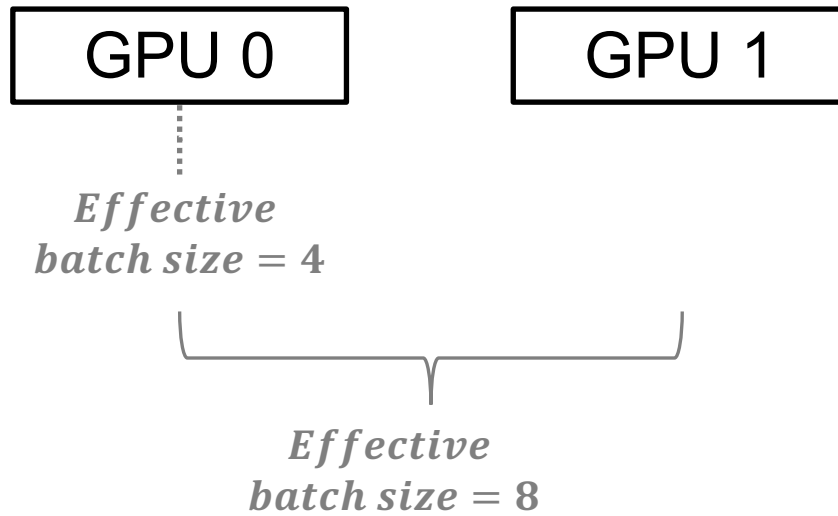
Specified batch size: 4



Distributed training

Effective batch size

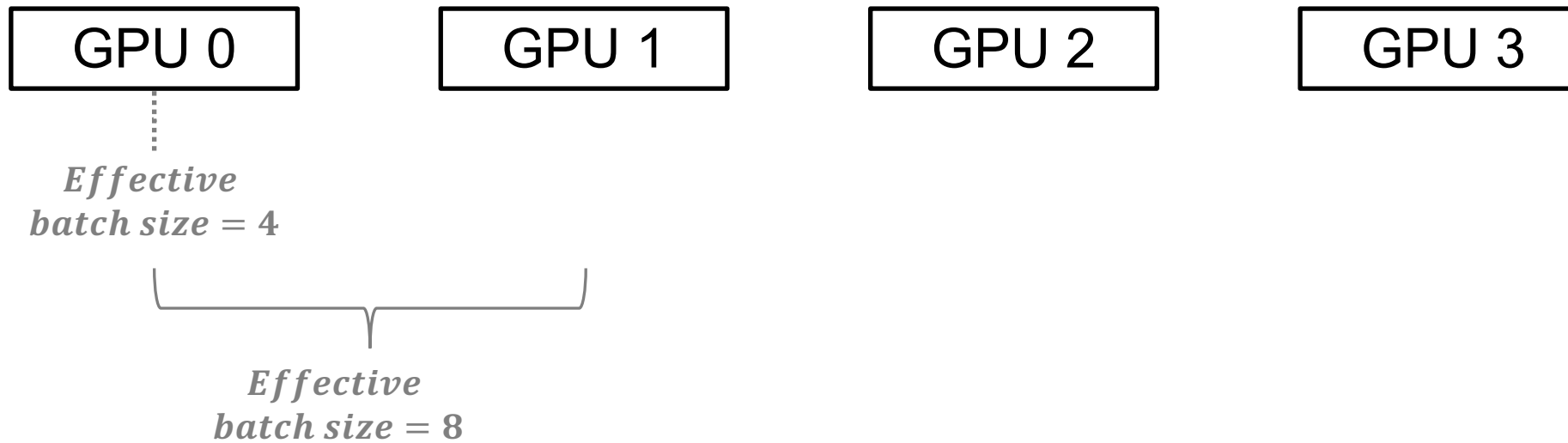
Specified batch size: 4



Distributed training

Effective batch size

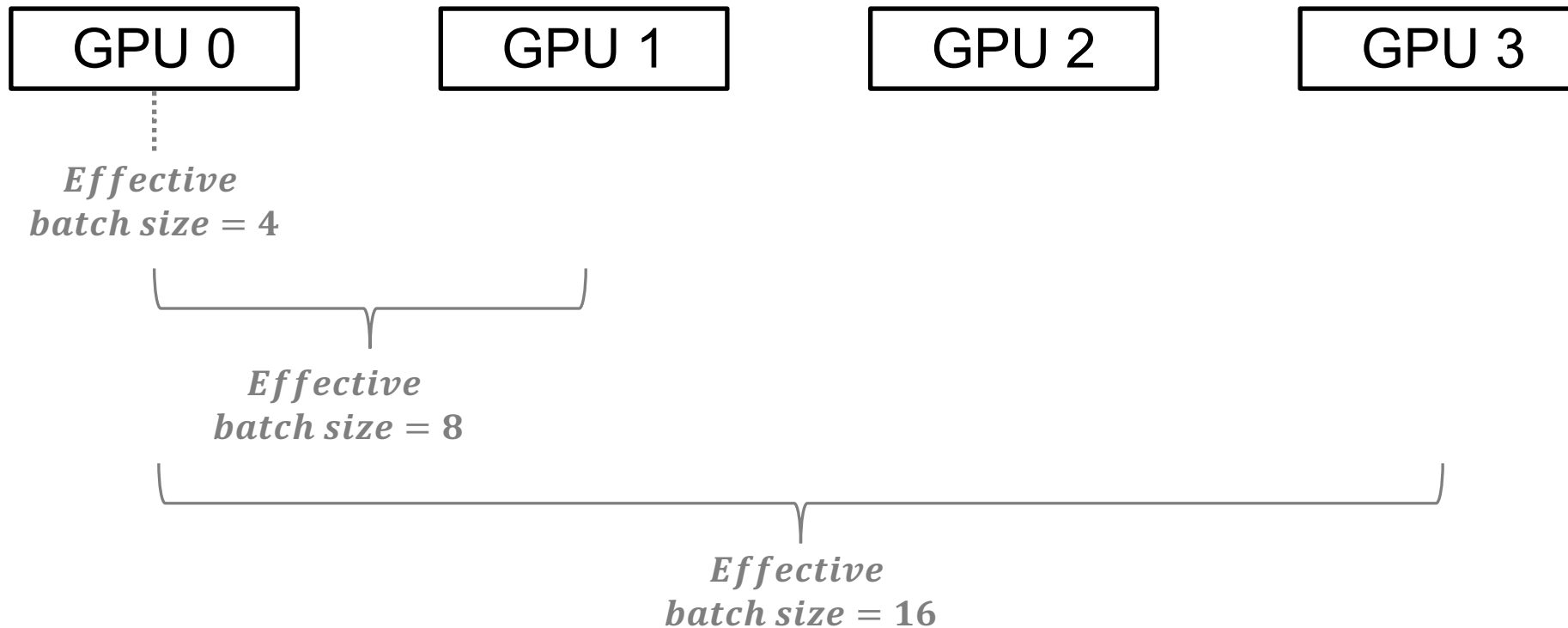
Specified batch size: 4



Distributed training

Effective batch size

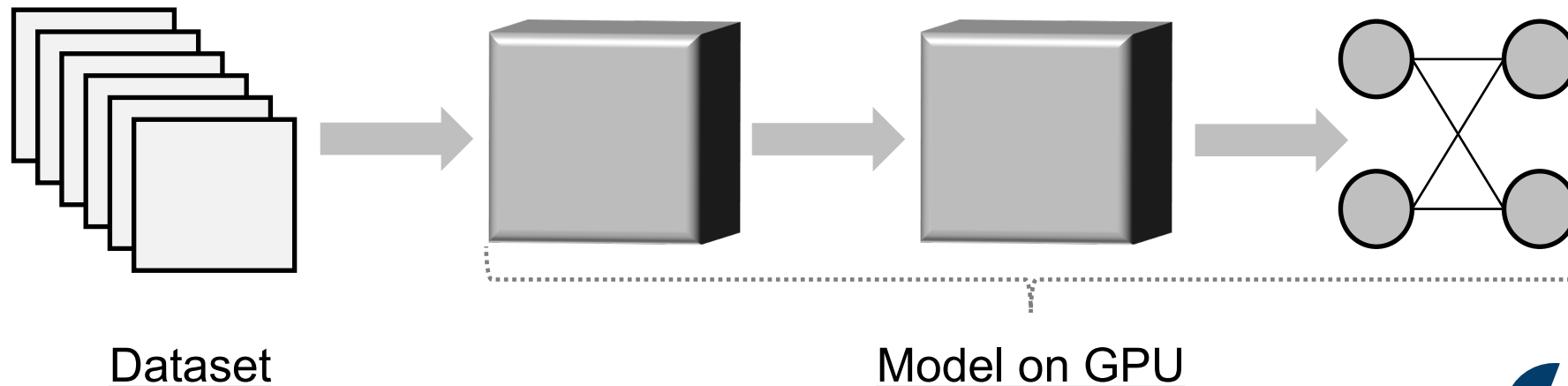
Specified batch size: 4



Distributed training

Motivation

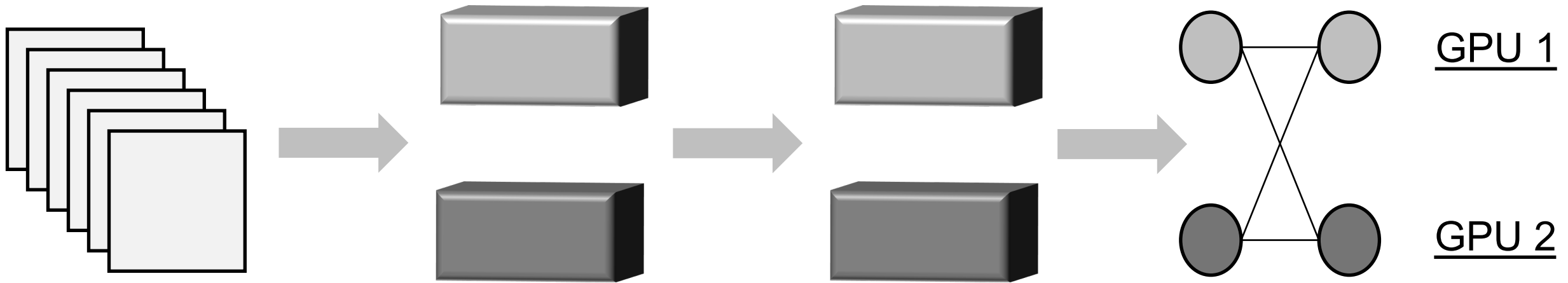
1. Train faster
 - i. Large dataset size
 - ii. Compute intensive model
2. Use a dataset with very large instances
3. Use a very large model
4. Increase the effective batch size



Distributed training

Model Parallel

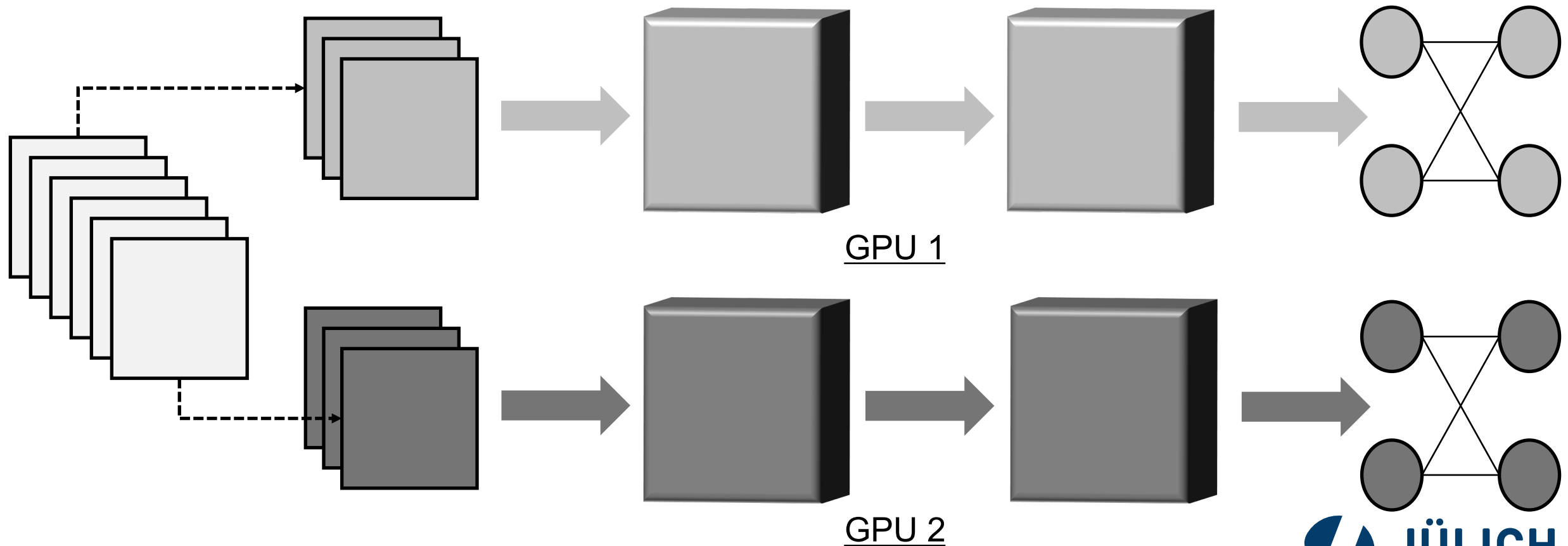
Ben-Nun, T., and Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR abs/1802.09941 (Feb. 2018)



Distributed training

Data Parallel

Ben-Nun, T., and Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR abs/1802.09941 (Feb. 2018)



Distributed training

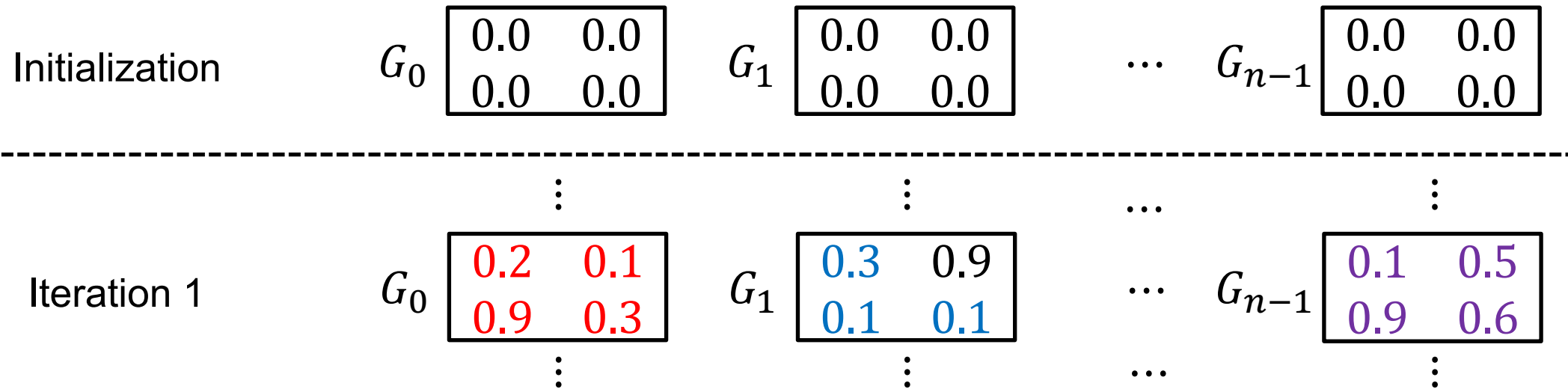
Gradient averaging in data parallel training

Initialization

$$G_0 \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad G_1 \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad \dots \quad G_{n-1} \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

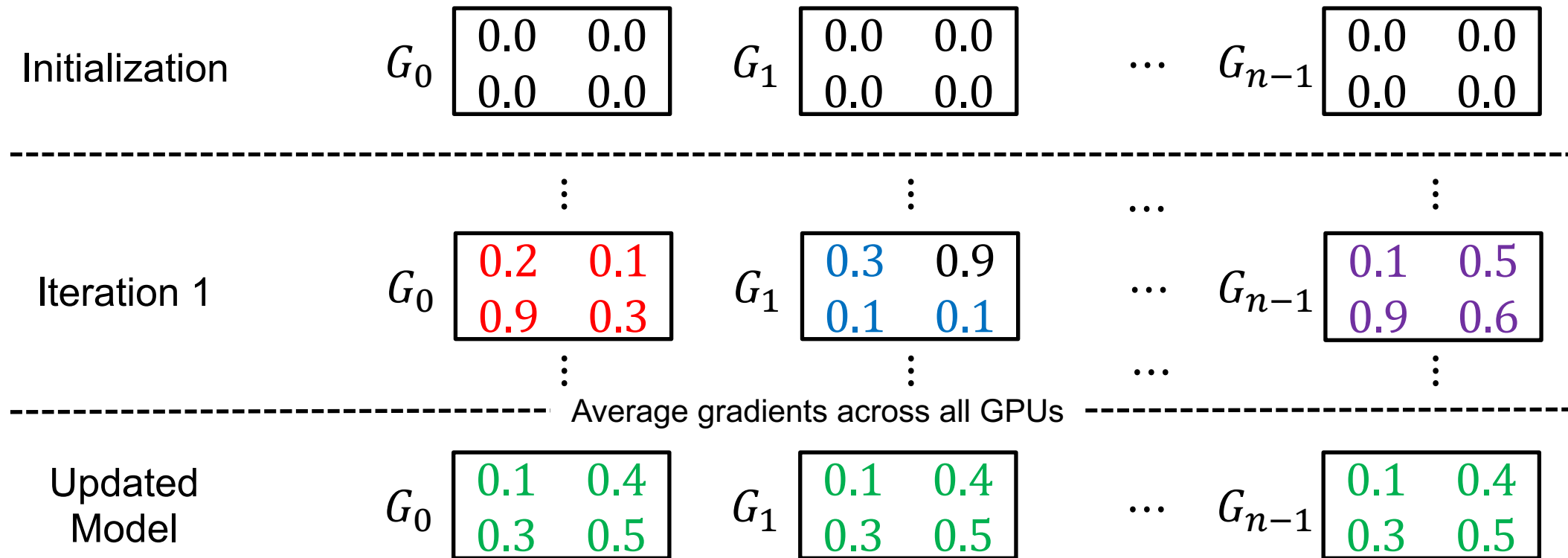
Distributed training

Gradient averaging in data parallel training



Distributed training

Gradient averaging in data parallel training



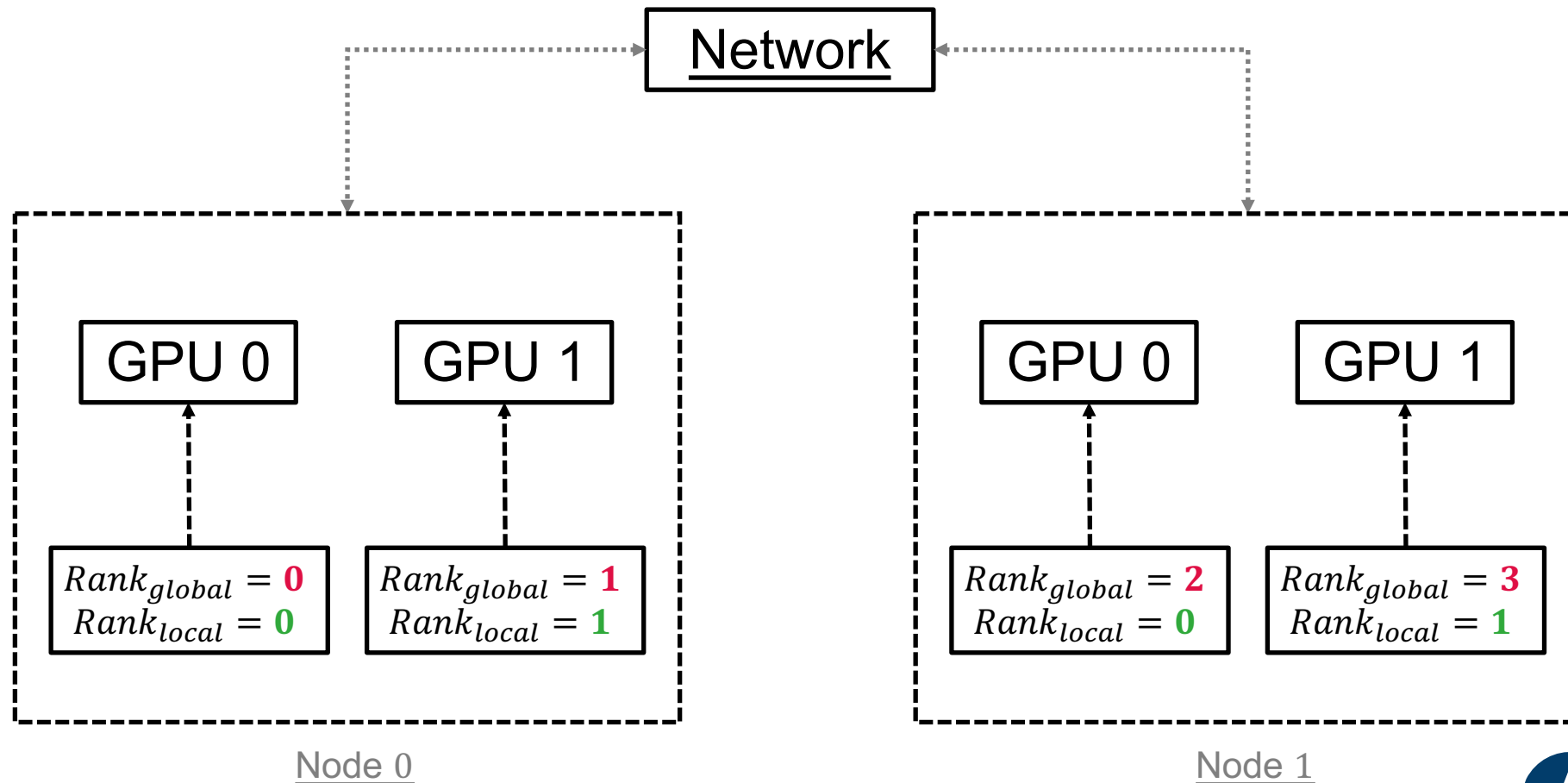
Agenda

1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

Distributed training

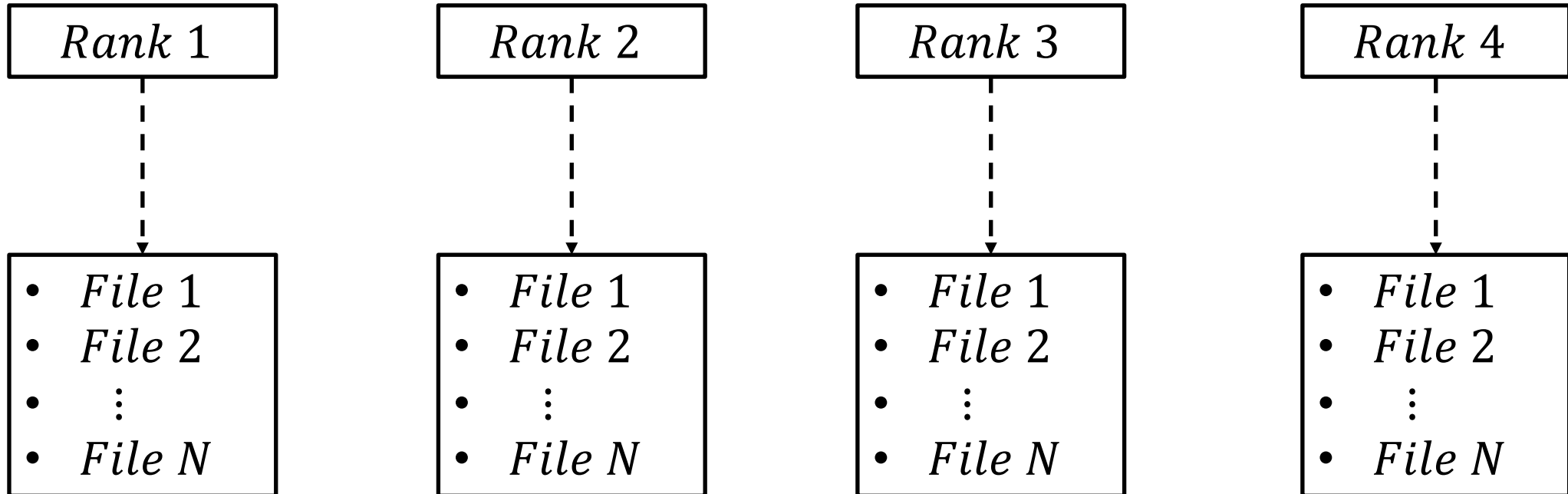
Global and Local ranks

```
export CUDA_VISIBLE_DEVICES=0,1
```



Distributed training

Epoch-wise distribution



- Each rank processes all files during an epoch
- Therefore, a single epoch on 4 GPUs is essentially equal to running 4 epochs

Distributed training: Examples

Distributed training with Horovod: Code (1)

```
1. import tensorflow as tf
```

Single GPU

```
1. import tensorflow as tf  
2. import math  
3. import horovod.tensorflow.keras as hvd  
4. hvd.init()  
5. gpus = tf.config.experimental.list_physical_devices('GPU')  
6. tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

Distributed

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`

Distributed training: Examples

Distributed training with Horovod: Code (2)

```
2. mnist = tf.keras.datasets.mnist
3. (x_train, y_train), (x_test, y_test) = mnist.load_data()
4. x_train, x_test = x_train / 255.0, x_test / 255.0
5. model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
6. optimizer = tf.keras.optimizers.Adam()
7. epochs = 4
```

Single GPU

```
7. mnist = tf.keras.datasets.mnist
8. (x_train, y_train), (x_test, y_test) = mnist.load_data()
9. x_train, x_test = x_train / 255.0, x_test / 255.0
10. model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
11. optimizer = tf.keras.optimizers.Adam()
12. optimizer = hvd.DistributedOptimizer(optimizer)
13. epochs = int(math.ceil(4.0 / hvd.size()))
```

Distributed

Distributed training: Examples

Distributed training with Horovod: Code (3)

```
8. model.compile(  
    optimizer=optimizer,  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy']  
)  
  
9. model.fit(  
    x=x_train,  
    y=y_train,  
    batch_size=32,  
    epochs=epochs  
)
```

Single GPU

```
14. model.compile(  
    optimizer=optimizer,  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy']  
)  
  
15. callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]  
16. model.fit(  
    x=x_train,  
    y=y_train,  
    batch_size=32,  
    epochs=epochs,  
    verbose=1 if hvd.rank() == 0 else 0,  
    callbacks=callbacks  
)
```

Distributed

Distributed training: Examples

Distributed training with Horovod: Output

```
$ mpirun -np 4 python -u mnist_epoch_distributed.py
```

```
Epoch 1/1
```

```
1875/1875 [=====] - 8s 125us/sample - loss: 0.2004 - acc: 0.9410
```

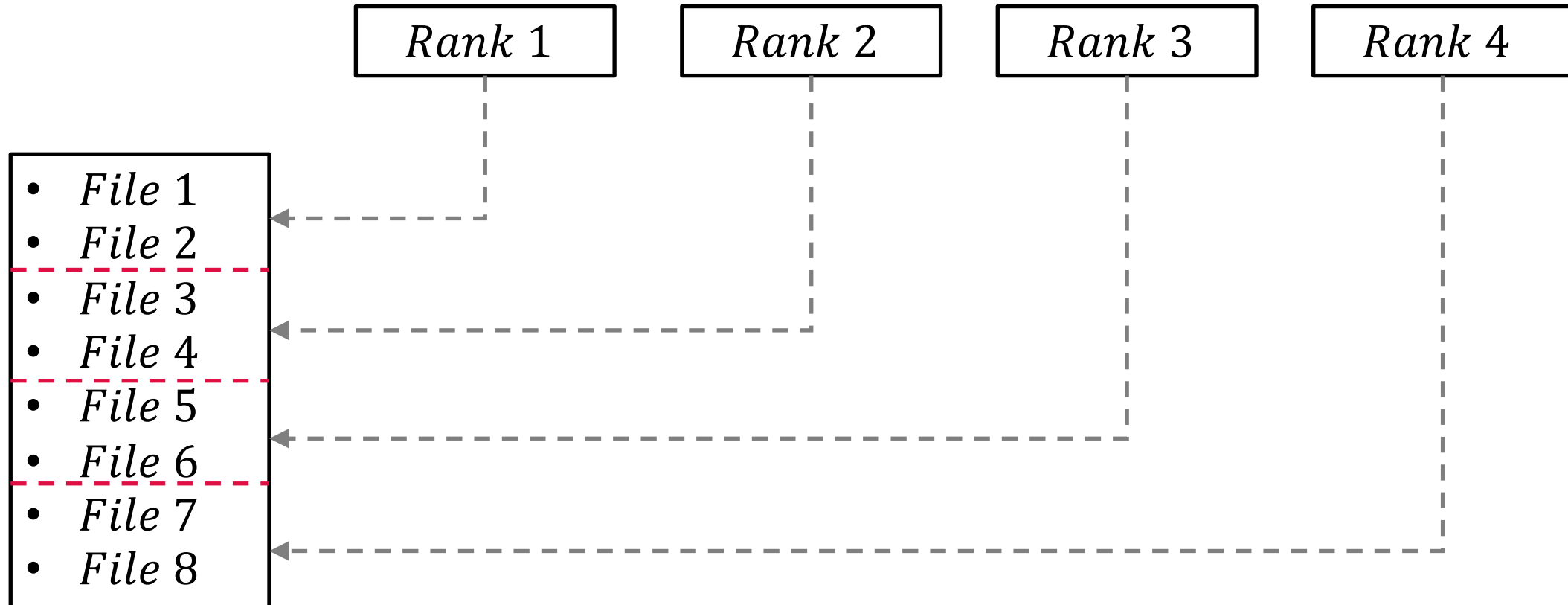
```
Test loss: 0.0761936013394734
```

```
Test accuracy: 0.9773
```

Distributed training

Distributing input data, i.e., instances, instead of epochs

- Consider a dataset with 8 input files, e.g., 8 images



Distributed training: Examples

Input data distribution for Horovod: Code snippet

```
1. mnist = tf.keras.datasets.mnist  
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```


Distributed training: Examples

Input data distribution for Horovod: Code snippet

```
1.  
2.  
3.  
4.  
5. def get_filenames(path):  
6.     return os.listdir(path)
```

```
7. ...  
8.  
9.  
10.  
11.  
12.  
13.  
14.  
15. ...
```

```
1. mnist = tf.keras.datasets.mnist  
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

dl_on_supercomputers/horovod_data_distributed/mnist_data_distributed.py

Distributed training: Examples

Input data distribution for Horovod: Code snippet

```
1.  
2.  
3.  
4. @DataDistributor(MPI.COMM_WORLD, shutdown_on_error=True)  
5. def get_filenames(path):  
6.     return os.listdir(path)
```

```
7. ...
```

```
8.
```

```
9.
```

```
10.
```

```
11.
```

```
12.
```

```
13.
```

```
14.
```

```
15. ...
```

```
1. mnist = tf.keras.datasets.mnist  
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

dl_on_supercomputers/horovod_data_distributed/mnist_data_distributed.py

Distributed training: Examples

Input data distribution for Horovod: Code snippet

```
1. from mpi4py import MPI
2. from hpc4neuro.distribution import DataDistributor
3.
4. @DataDistributor(MPI.COMM_WORLD, shutdown_on_error=True)
5. def get_filenames(path):
6.     return os.listdir(path)
```

```
7. ...
```

```
8.
```

```
9.
```

```
10.
```

```
11.
```

```
12.
```

```
13.
```

```
14.
```

```
15. ...
```

```
1. mnist = tf.keras.datasets.mnist
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

dl_on_supercomputers/horovod_data_distributed/mnist_data_distributed.py

Distributed training: Examples

Input data distribution for Horovod: Code snippet

```
1. from mpi4py import MPI
2. from hpc4neuro.distribution import DataDistributor
3.
4. @DataDistributor(MPI.COMM_WORLD, shutdown_on_error=True)
5. def get_filenames(path):
6.     return os.listdir(path)
7. ...
8.
9. data_dir = 'data/mnist/partitioned'
10. train_filenames = get_filenames(f'{data_dir}/train')
11.
12. x_train, y_train = load_dataset(f'{data_dir}/train', train_filenames)
13.
14. x_train = x_train / 255.0
15. ...
```

```
1. mnist = tf.keras.datasets.mnist
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

dl_on_supercomputers/horovod_data_distributed/mnist_data_distributed.py

Agenda

1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Horovod ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

Deep learning on Supercomputers

The tutorial

The screenshot shows the GitLab web interface for the project 'dl_on_supercomputers' under the namespace 'HPC4NS'. The project ID is 2346. It has 54 commits, 8 branches, 0 tags, 155.7 MB of files, and 155.8 MB of storage. The description states: 'Samples and documentation for the "Getting started with Deep Learning on Supercomputers" tutorial.' The interface shows the 'master' branch selected. A recent merge of branch 'tf2' into 'master' by Fahad Khalid is shown, committed 37 minutes ago with hash cea58c89. Below the merge, there are tabs for 'README' and 'Other'. A table lists the files in the repository, all of which were updated to use TensorFlow2 37 minutes ago.

Name	Last commit	Last update
course_material	Updated to use Tensorflow2	37 minutes ago
datasets	Updated to use Tensorflow2	37 minutes ago
tensorflow	Updated to use Tensorflow2	37 minutes ago
training_data_distribution	Updated to use Tensorflow2	37 minutes ago
utils	Updated to use Tensorflow2	37 minutes ago
.gitattributes	Updated to use Tensorflow2	37 minutes ago
.gitignore	Updated to use Tensorflow2	37 minutes ago

Deep learning on Supercomputers

The tutorial

Name
course_material
datasets
tensorflow
training_data_distribution
utils
.gitattributes
.gitignore
LICENSE
NOTICE
README.md
requirements.txt

Table of contents

1. [A word regarding the code samples](#)
2. [Changes made to support loading of pre-downloaded datasets](#)
3. [Applying for user accounts on supercomputers](#)
4. [Logging on to the supercomputers](#)
5. [Cloning the repository](#)
6. [Running a sample](#)
7. [Distributed training](#)
8. [Credits](#)

Deep learning on Supercomputers

The tutorial: Job script jureca_job.sh

```
1. # Slurm job configuration
2. #SBATCH --nodes=2
3. #SBATCH --ntasks=8
4. #SBATCH --ntasks-per-node=4
5. #SBATCH --output=output_%j.out
6. #SBATCH --error=error_%j.er
7. #SBATCH --time=00:10:00
8. #SBATCH --job-name=TUTORIAL
9. #SBATCH --gres=gpu:4 --partition=dc-gpu-devel
10. #SBATCH --mail-type=ALL
11. #SBATCH --mail-user=<...>
```

```
12. # Load the required modules
13. module load GCC/9.3.0
14. module load OpenMPI/4.1.0rc1
15. module load TensorFlow/2.3.1-Python-3.8.5
16. module load Horovod/0.20.3-Python-3.8.5
17.
18. # Make all GPUs visible per node
19. export CUDA_VISIBLE_DEVICES=0,1,2,3
20.
21. # Run the program
22. srun python -u keras_mnist.py
```


Deep learning on Supercomputers

The tutorial: Starting and monitoring the training job

```
[username@jrlogin06 jureca-dc]$ cd dl_on_supercomputers/tensorflow/
[username@jrlogin06 tensorflow]$ sbatch jureca_job.sh
Submitted batch job 9092135
[username@jrlogin06 tensorflow]$ squeue -u username
      JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
      9092135 dc-gpu-de TUTORIAL username  R        0:14        1 [jrc0438,jrc0439]
[username@jrlogin06 tensorflow]$ tail -f output_9092135.out
...
Epoch 1/10
125/125 [=====] - 1s 5ms/step - loss: 0.3714 - accuracy: 0.8845
Epoch 2/10
125/125 [=====] - 1s 5ms/step - loss: 0.0929 - accuracy: 0.9711
Epoch 3/10
121/125 [=====>.] - ETA: 0s - loss: 0.0752 - accuracy: 0.9780
...
Epoch 10/10
125/125 [=====] - 1s 7ms/step - loss: 0.0283 - accuracy: 0.9909
```

Agenda

1. Code example: Training with one GPU
 - i. Handwritten digit recognition
 - ii. The MNIST dataset
 - iii. Implementation with `tf.keras`
2. Concepts: Distributed training
 - i. Why use distributed training?
 - ii. Model parallelism
 - iii. Data parallelism
 - iv. Gradient aggregation
3. Code examples: Distributed training
 - i. Horovod ranks: Global vs. Local
 - ii. MNIST classification: Epoch distributed
 - iii. Distributing training data
4. The “Deep Learning on Supercomputers” tutorial
 - i. Introduction
 - ii. Job configuration
 - iii. Running code samples on JURECA
5. Conclusion

Conclusion

Summary

Key points

- There is more to distributed training than speedup, e.g., effective batch size can be increased
- The supercomputers can be utilized for data parallel training with relative ease
- Tensorflow and Horovod are already available on JURECA, JUWELS, JUWELS-Booster, and JUSUF
- Combining the material presented here with the “DL on Supercomputers” tutorial is a good place to start
- A good foundation in parallel programming, especially with MPI, can go a long way

Support

- All SC related issues: sc@fz-juelich.de
- Tutorial and hpc4neuro: slns@fz-juelich.de

Useful resources

1. Tutorial
 - https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers
2. The hpc4neuro Python library
 - <https://gitlab.version.fz-juelich.de/hpc4ns/hpc4neuro>

[Thank you!](#)