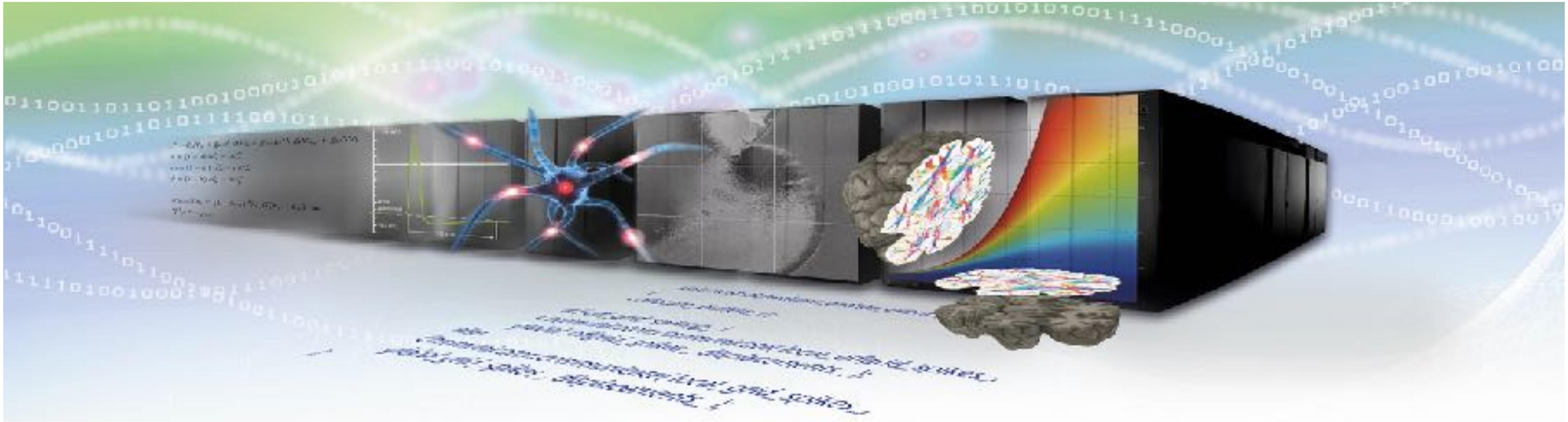


# Deep Learning on Supercomputers

## An introduction

Alexandre Strube [a.strube@fz-juelich.de]. Helmholtz AI, Jülich Supercomputing Centre.

November 2022. Course: Introduction to Supercomputing at JSC - Theory & Practice.



JÜLICH Forschungszentrum

# Agenda

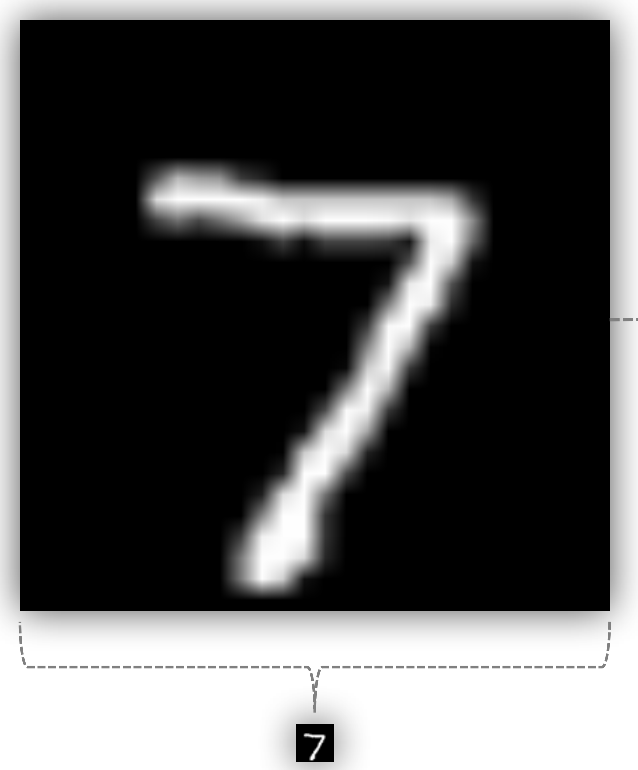
1. Code example: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with `tf.keras`
2. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation
3. Code examples: Distributed training
   i. MNIST classification: Epoch distributed
   ii. 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. MNIST classification: Epoch distributed
   ii. 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

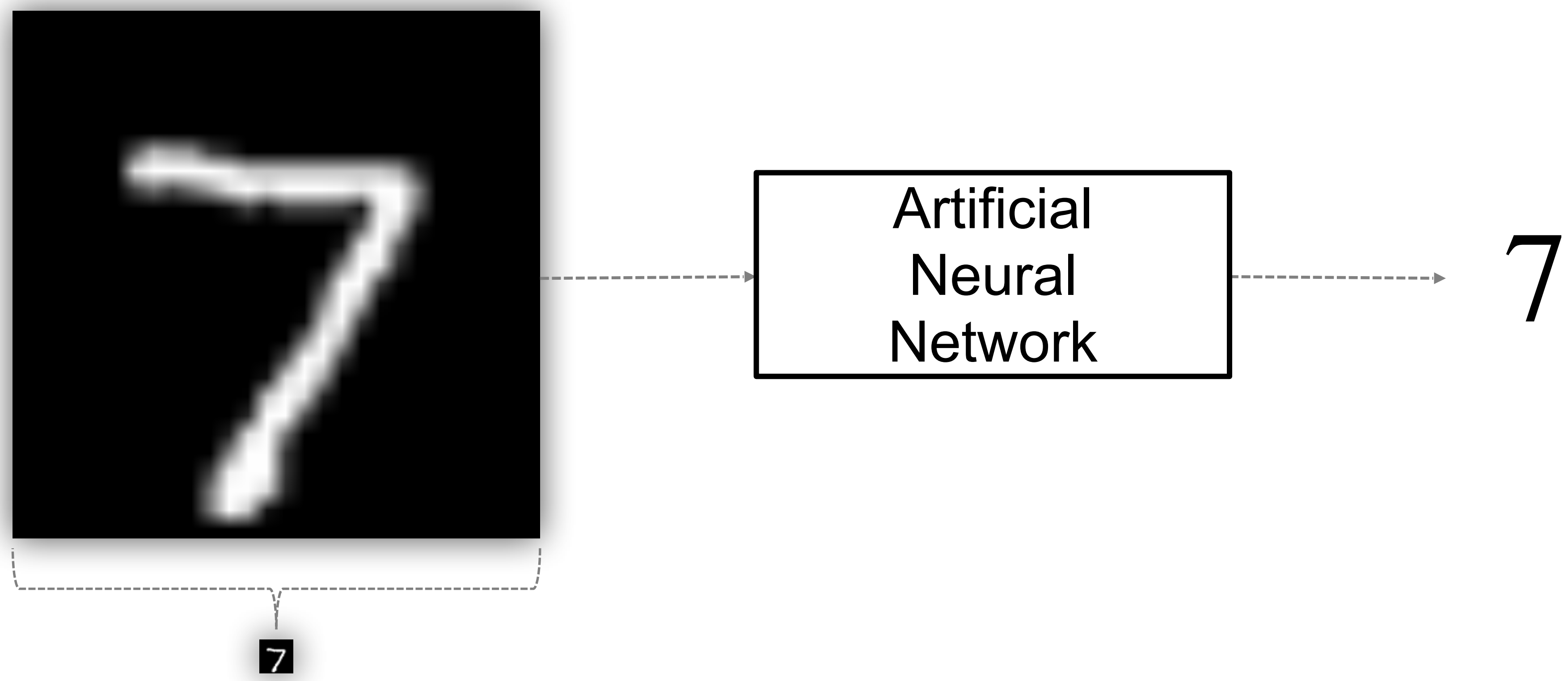


$f(image)$

7

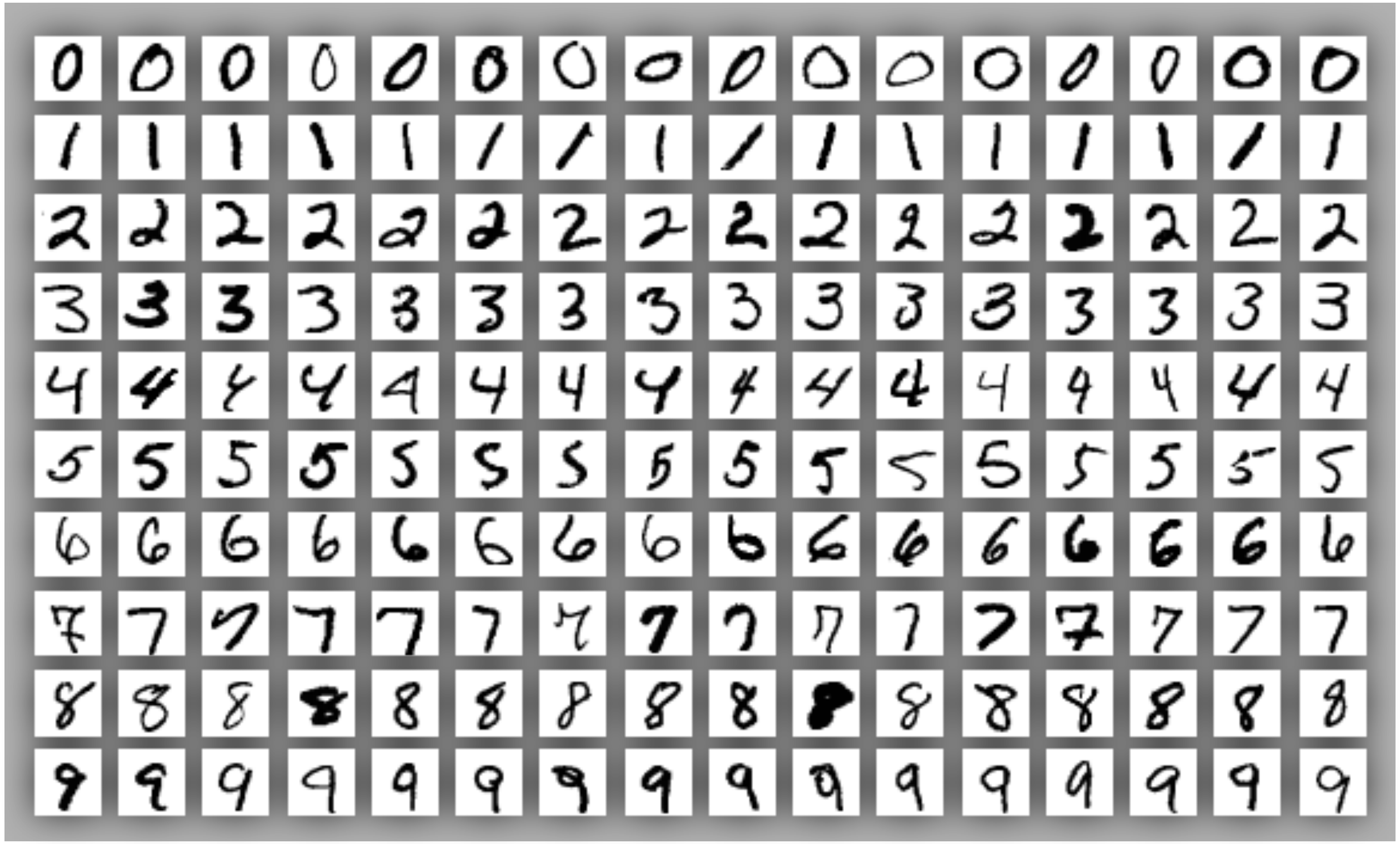
# Example

## Handwritten digit recognition



# Example

**Modified National Institute of Standards and Technology (MNIST) database**



Source: Modified version of this. License.

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



JÜLICH Forschungszentrum

# Example

## A basic network for classification

Image

# Example

## A basic network for classification

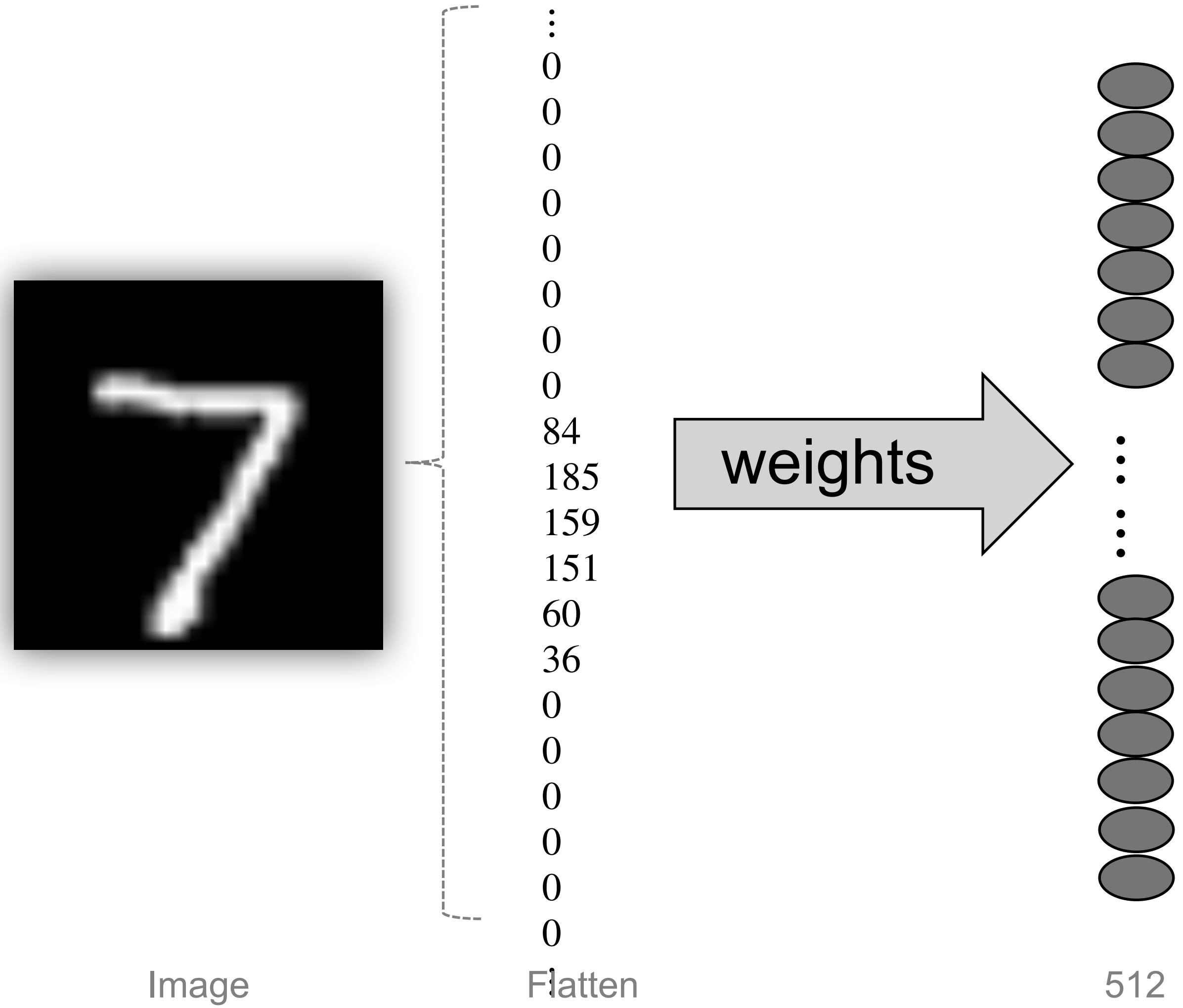
⋮
0
0
0
0
0
0
0
0
84
185
159
151
60
36
0
0
0
0
0
0
⋮

Image

Flatten

# Example

## A basic network for classification

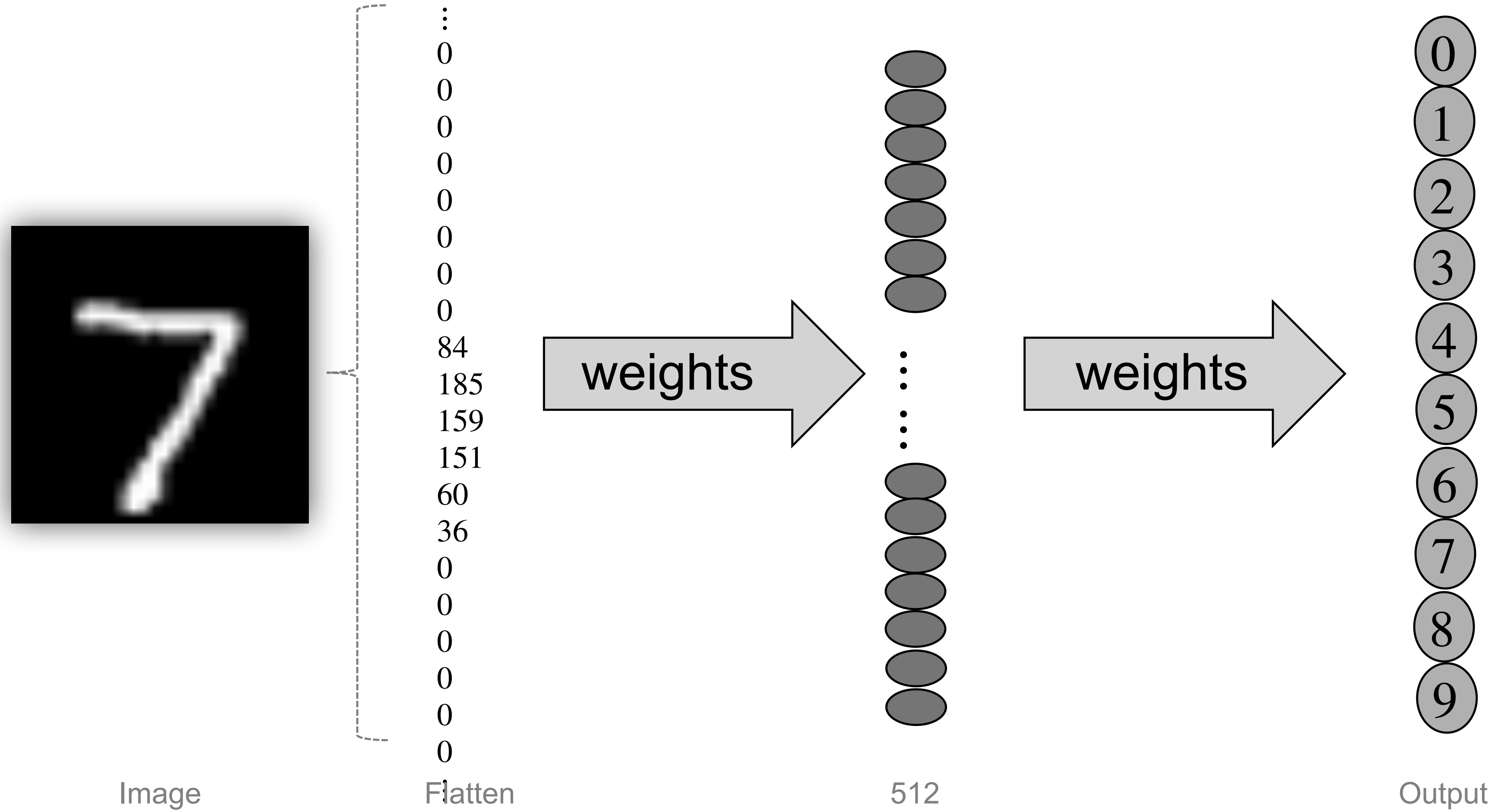





JÜLICH
Forschungszentrum

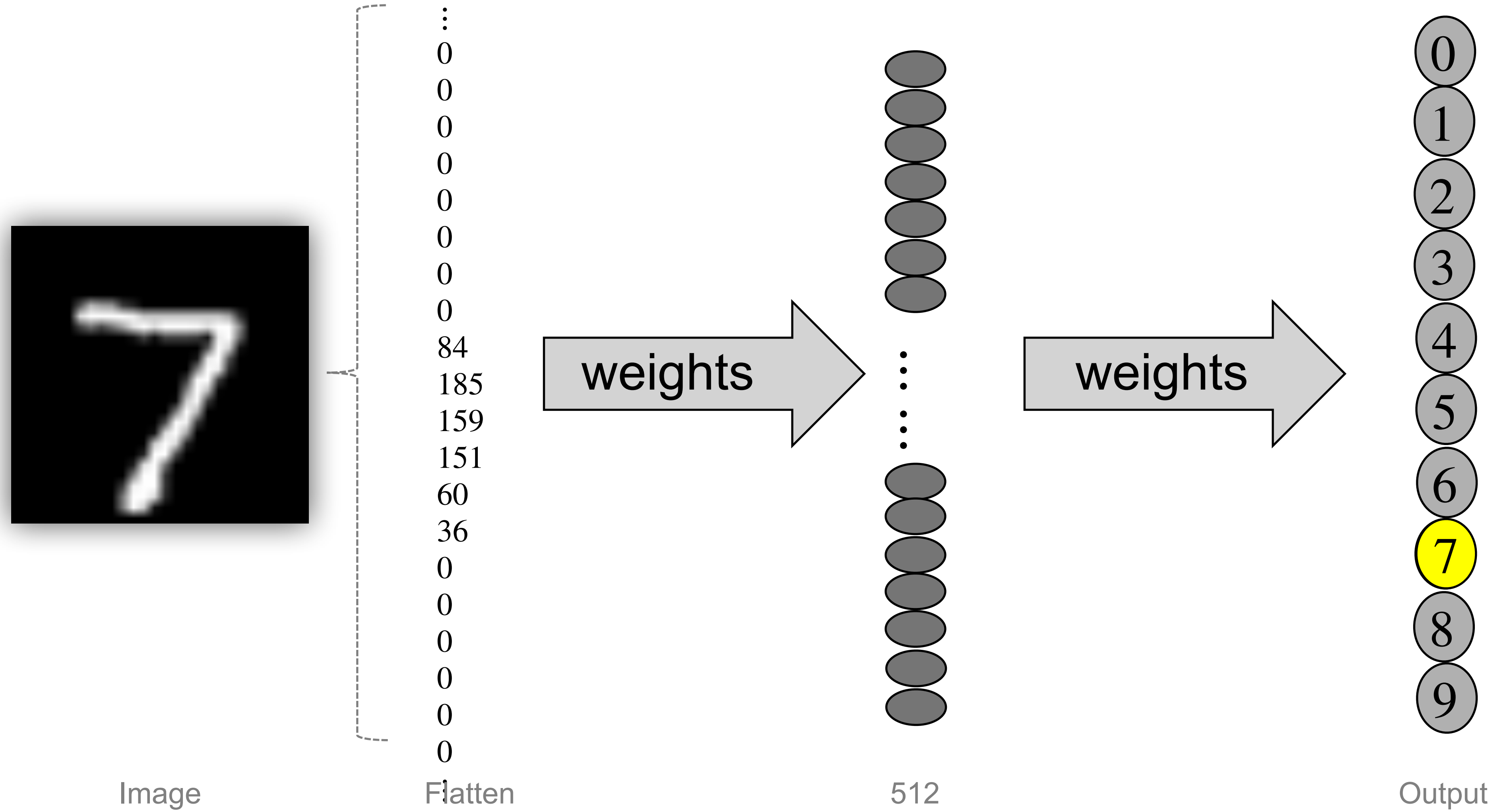
# Example

## A basic network for classification



# Example

## A basic network for classification



# Example

**MNIST classification with `tf.keras`: Code**



Forschungszentrum

# Example

## MNIST classification with `tf.keras`: Code

```
import tensorflow as tf
```

dl_on_supercomputers/course_material/examples/mnist_single_gpu.py

# Example

## MNIST classification with `tf.keras`: Code

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# Example

## MNIST classification with `tf.keras`: Code

```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# Example

## MNIST classification with `tf.keras`: Code

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

dl_on_supercomputers/course_material/examples/mnist_single_gpu.py

# Example

## MNIST classification with `tf.keras`: Code

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# 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()
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# Example

## MNIST classification with `tf.keras`: Code

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
 optimizer = tf.keras.optimizers.Adam()
 epochs = 4
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# 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']
    )
```

`dl_on_supercomputers/course_material/examples/mnist_single_gpu.py`

# 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`: 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**

# Example

## MNIST classification with `tf.keras`: Output

```
$ python -u 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
```

# 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
```

# 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
```

# 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
```

# 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
```

# 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
```

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


JÜLICH Forschungszentrum

# Example

## MNIST classification with `tf.keras`: Output

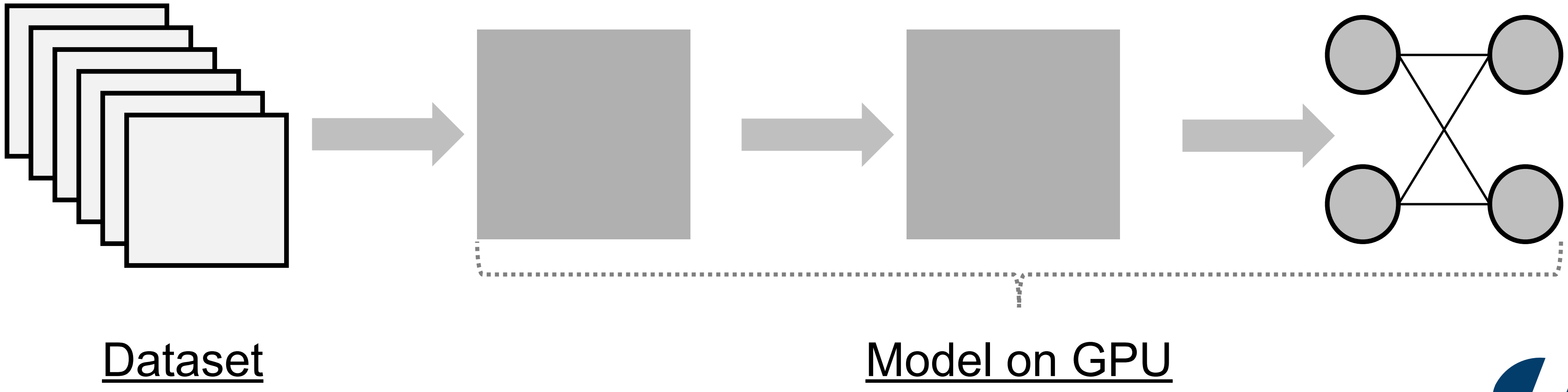
```
$ python -u mnist_single_gpu.py

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
```

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

# Agenda

# Distributed training

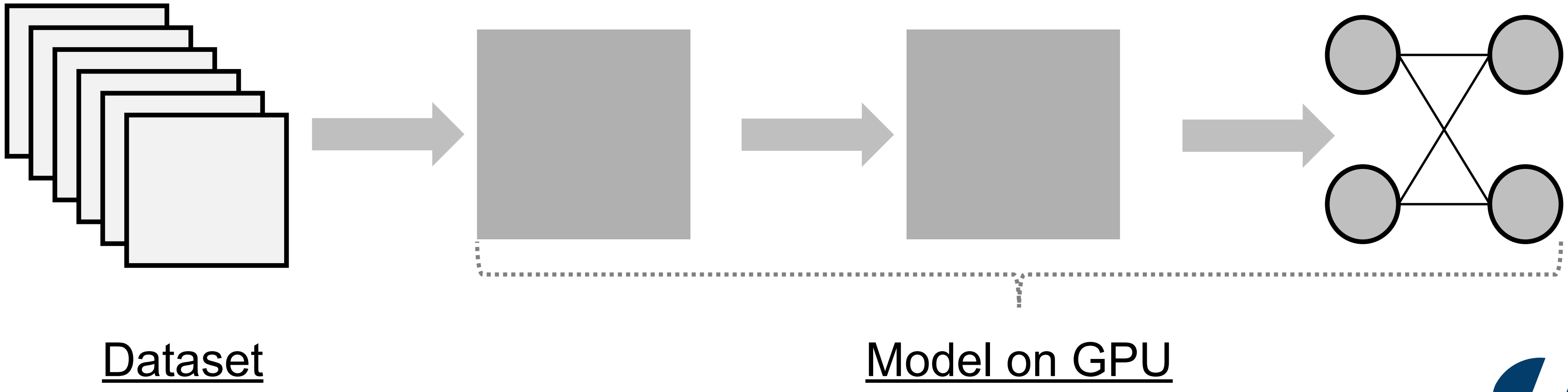
## Motivation



# Distributed training

### Motivation

1. Train faster
   i. Large dataset size
   ii. Compute intensive model



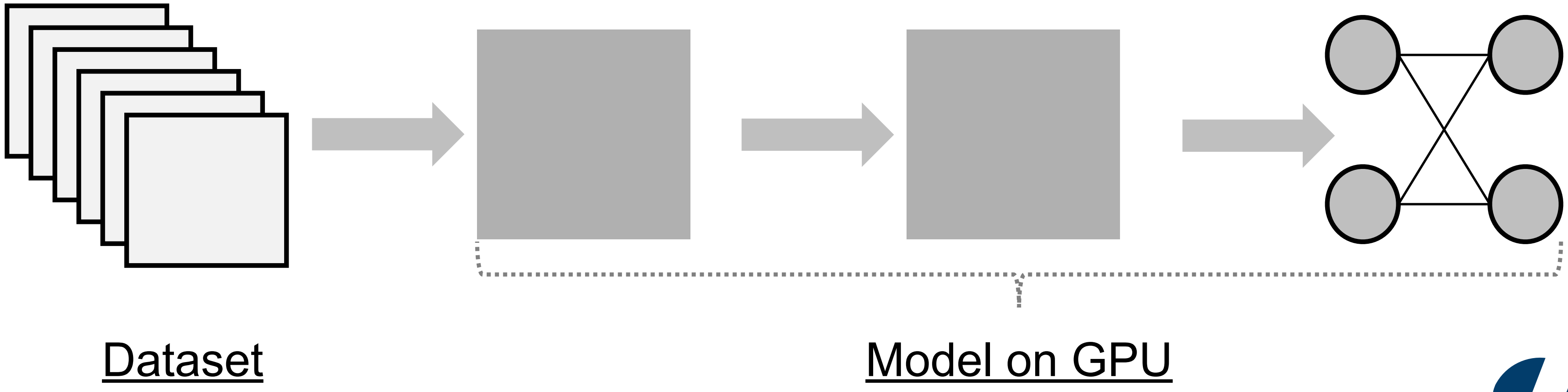



JÜLICH Forschungszentrum

# Distributed training

### Motivation

1. Train faster
   i. Large dataset size
   ii. Compute intensive model
2. Use a dataset in very large machines



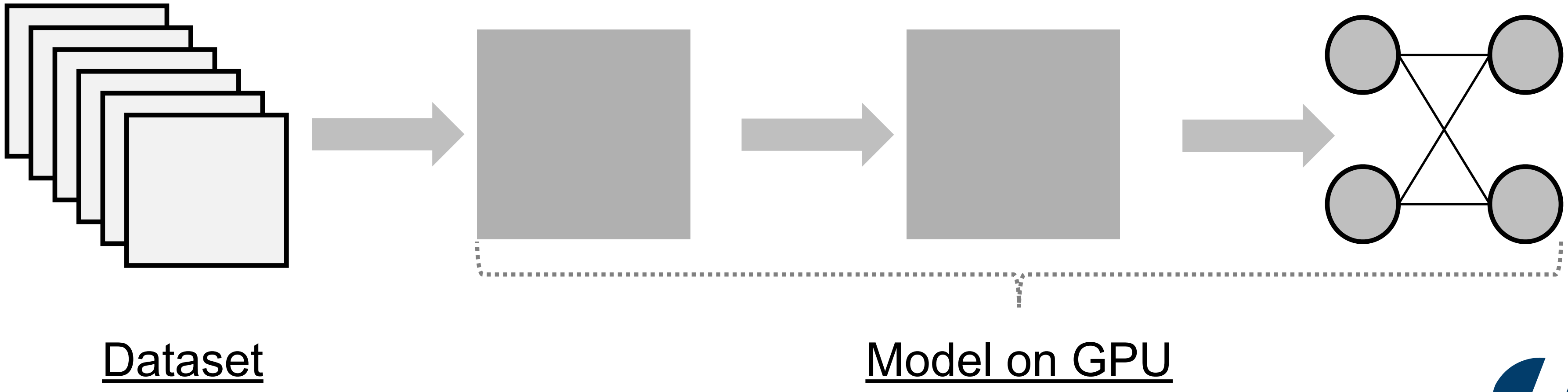



JÜLICH Forschungszentrum

# Distributed training

**Motivation**

1. Train faster
   i. Large dataset size
   ii. Compute intensive model
2. Use a dataset in very large machines
3. Use a very large model

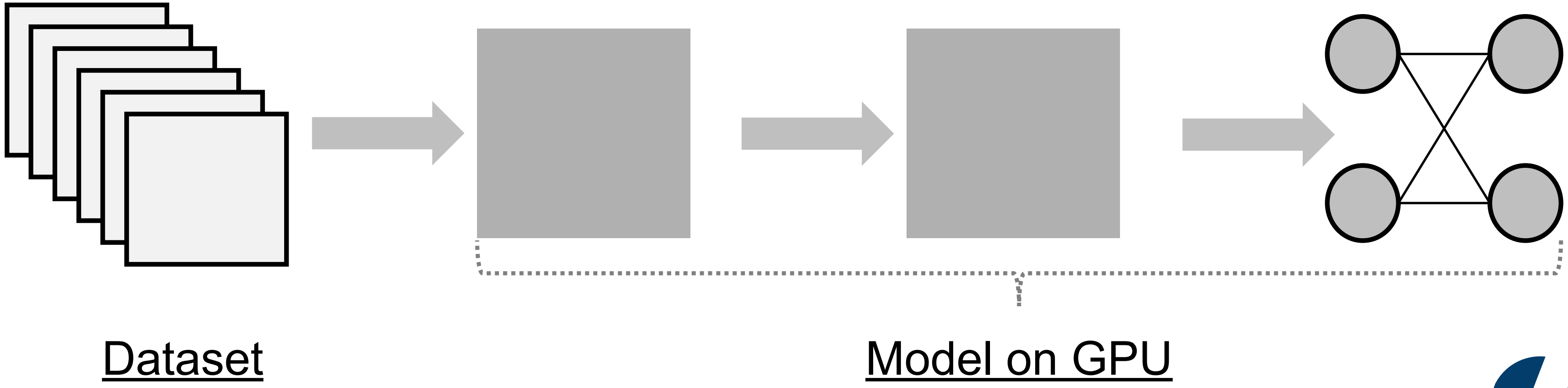


# Distributed training

## Motivation

1. Train faster
   i. Large dataset size
   ii. Compute intensive model
2. Use a dataset in very large machines
3. Use a very large model

4. Increase the effective batch size





JÜLICH Forschungszentrum

# 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`

GPU 0

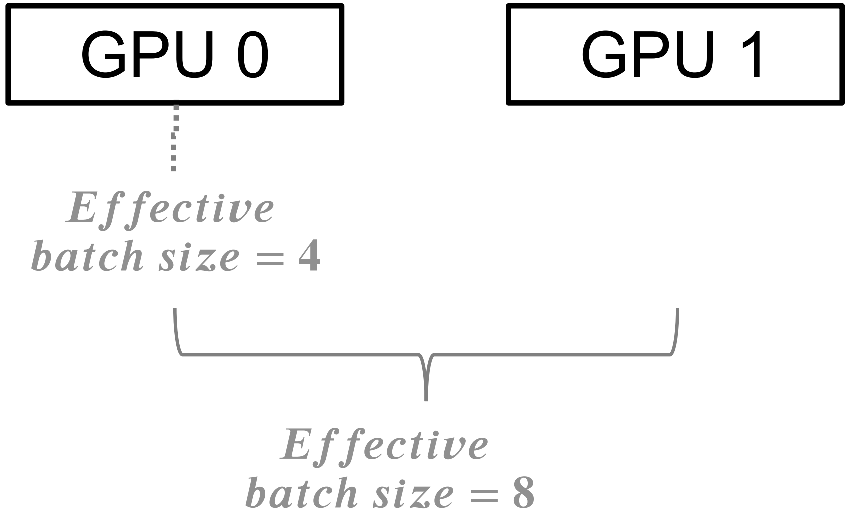
GPU 1

*Effective batch size* $= 4$

# Distributed training

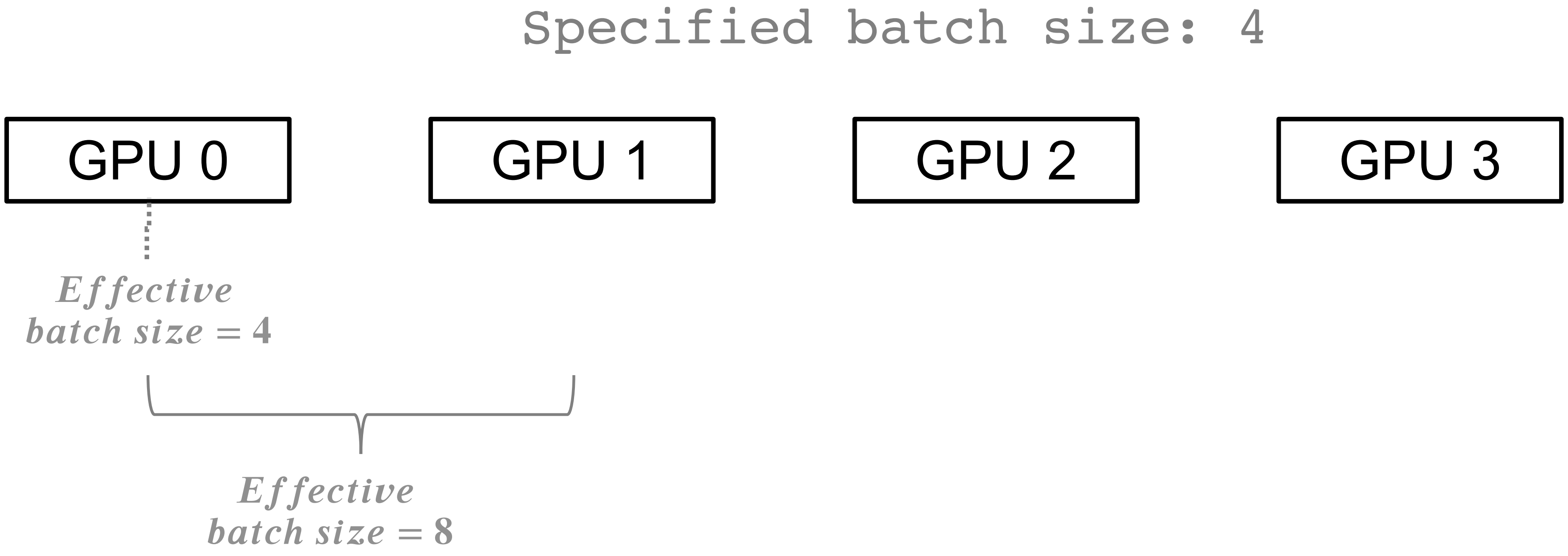
## Effective batch size

`Specified batch size: 4`



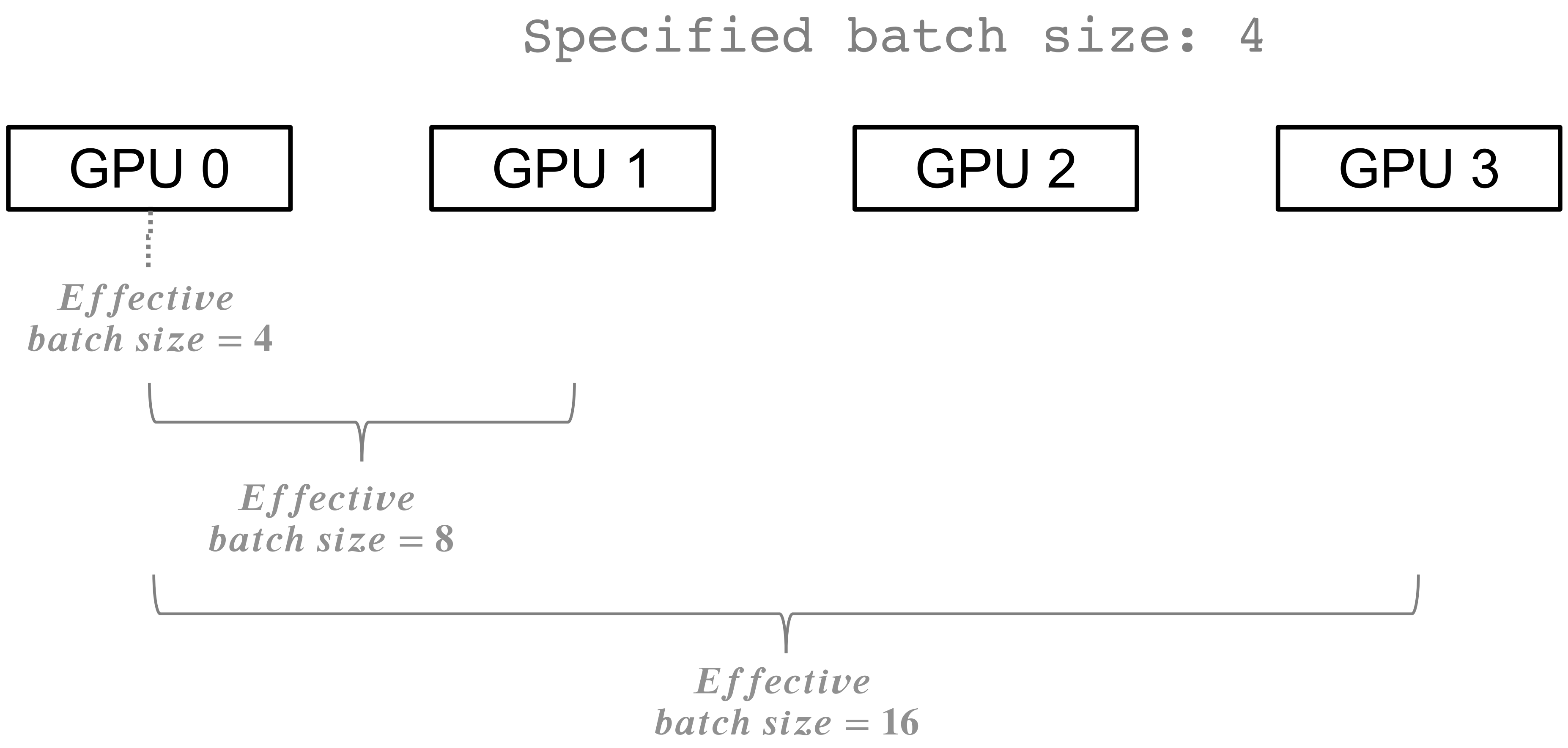
# Distributed training

## Effective batch size



# Distributed training

## Effective batch size

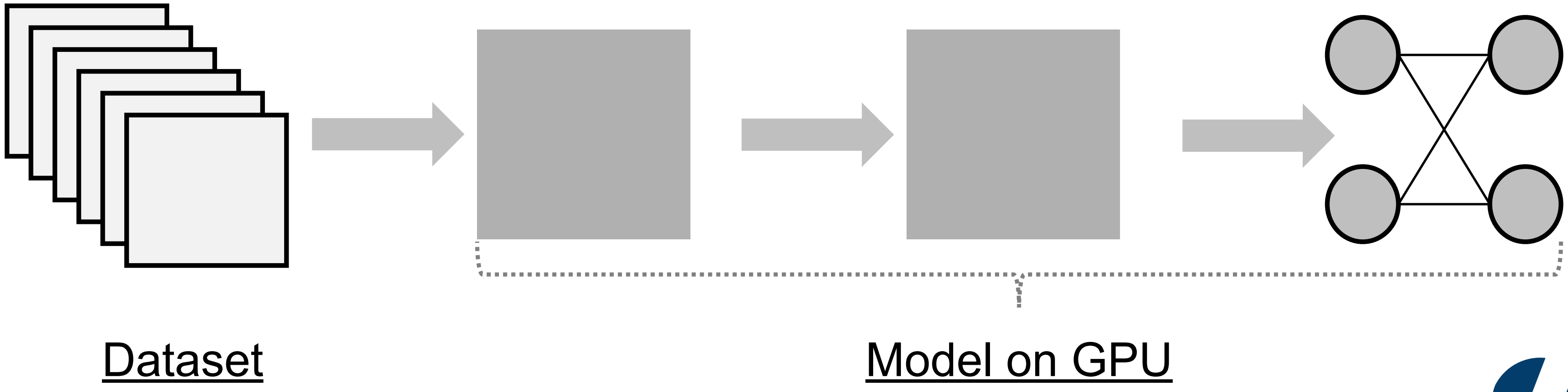


# Distributed training

## Motivation

1. Train faster
   i. Large dataset size
   ii. Compute intensive model
2. Use a dataset in very large machines
3. Use a very large model

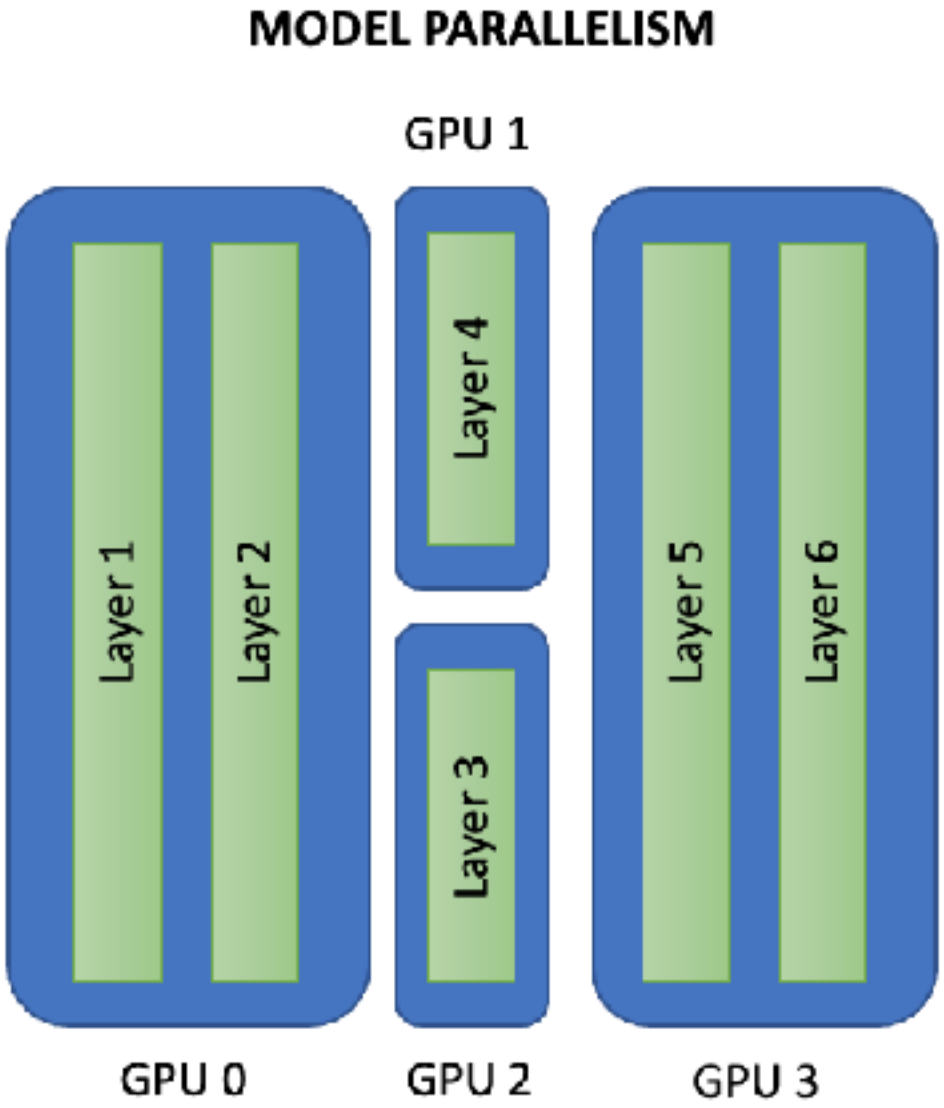
4. Increase the effective batch size



# Distributed training

## Model Parallel

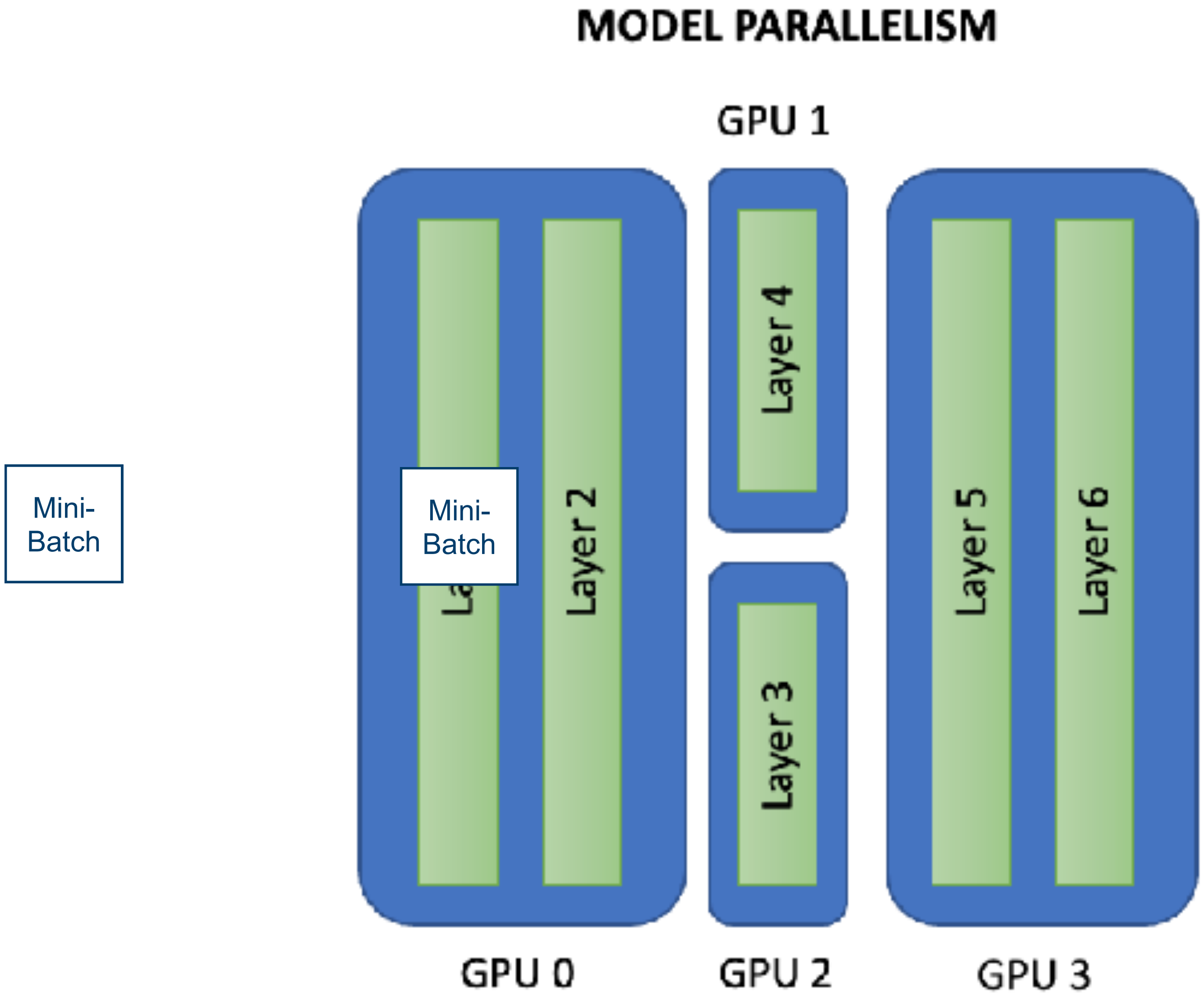
Mini-
Batch




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Model Parallel



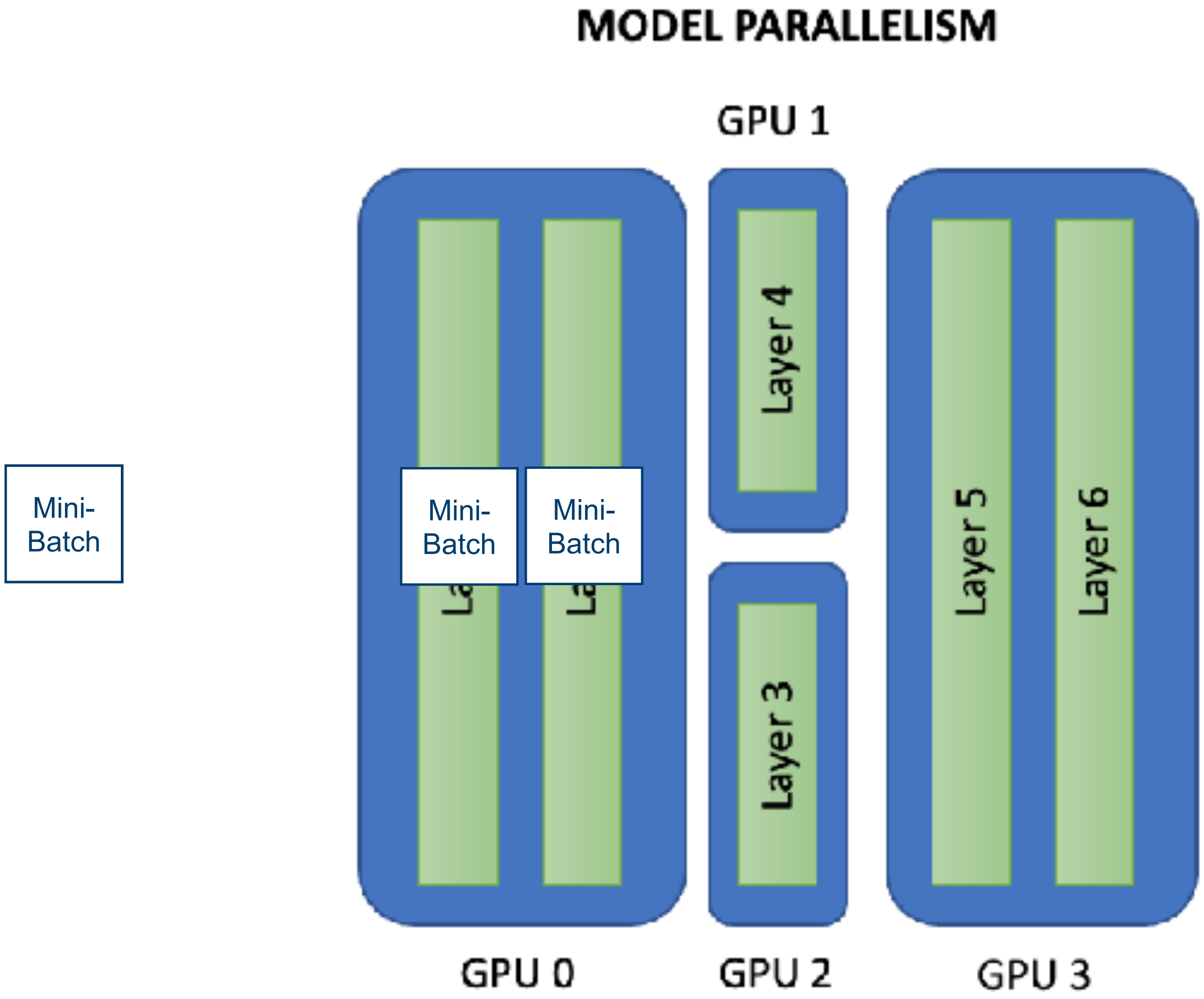

Jordi Torres.AI: Deep Learning on Supercomputers



JÜLICH Forschungszentrum

# Distributed training

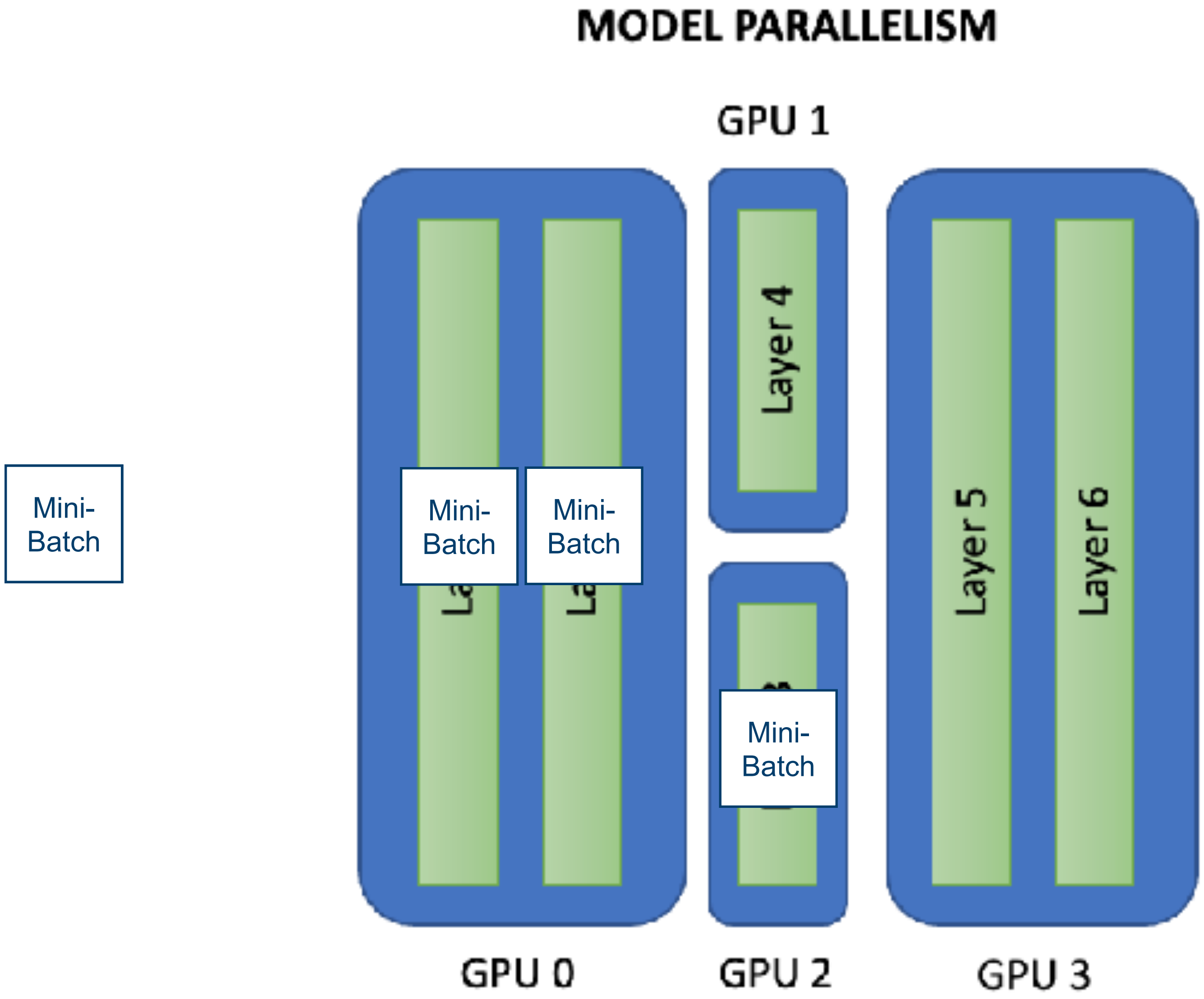
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

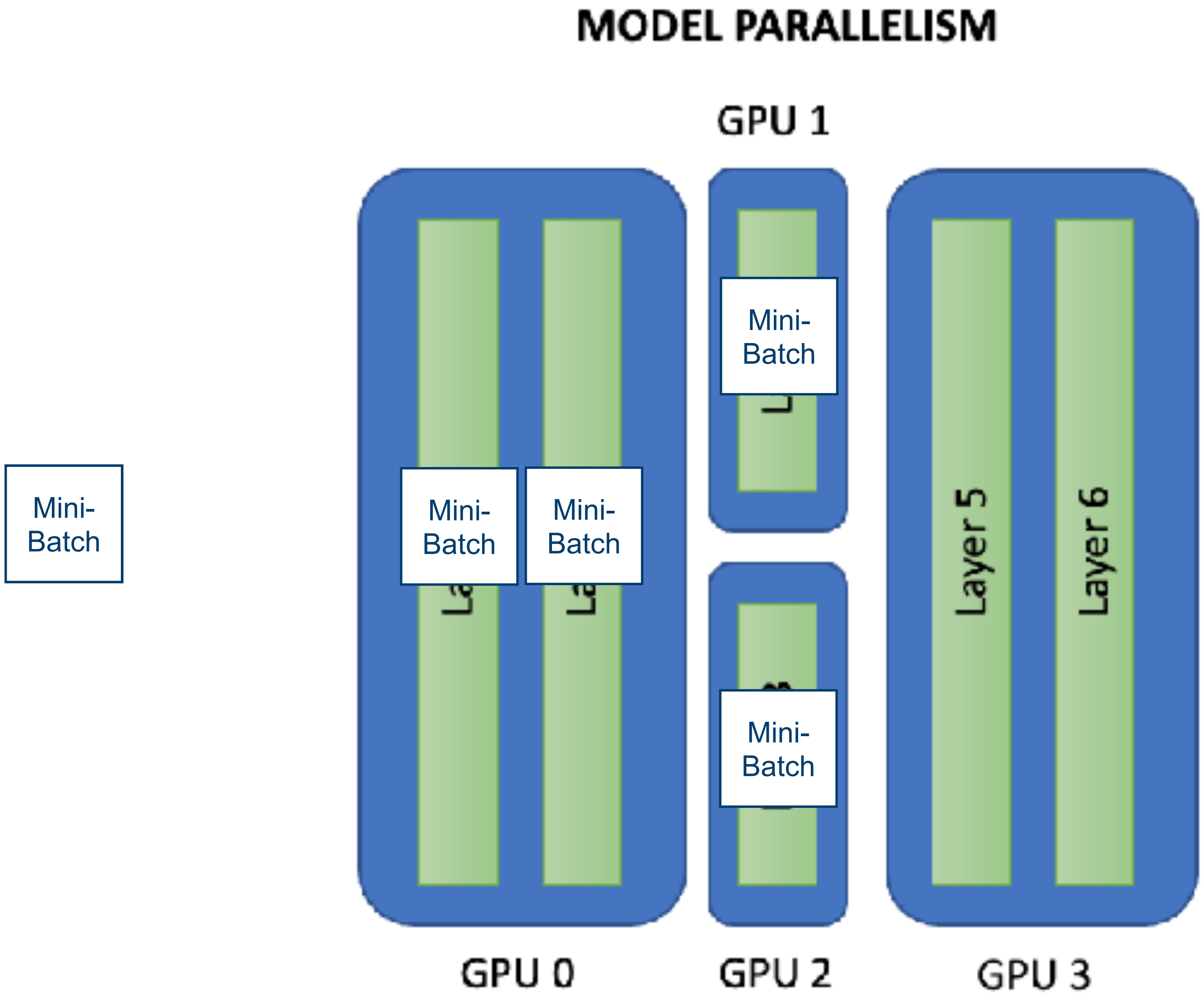
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Model Parallel

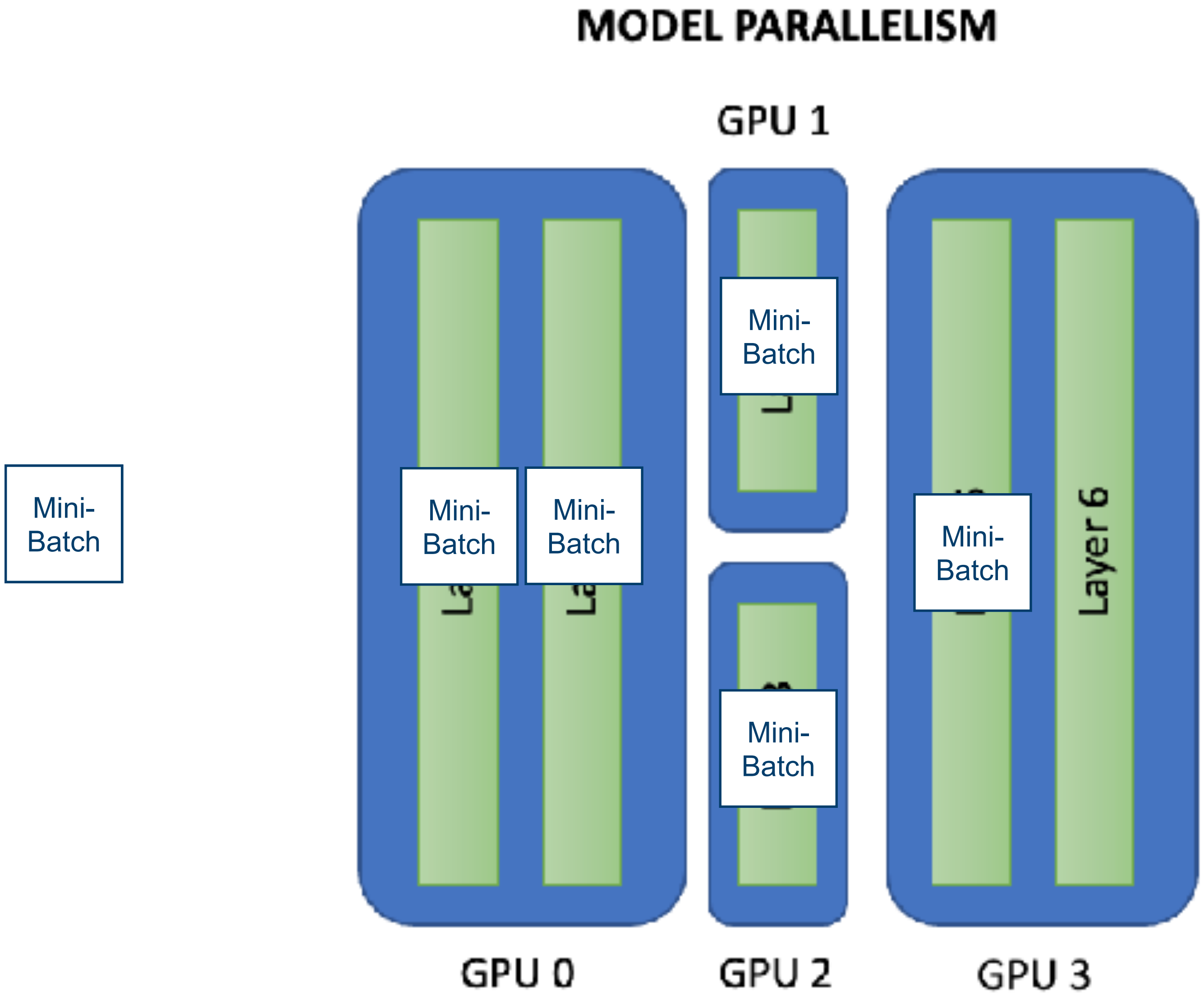



Jordi Torres.AI: Deep Learning on Supercomputers


JÜLICH Forschungszentrum

# Distributed training

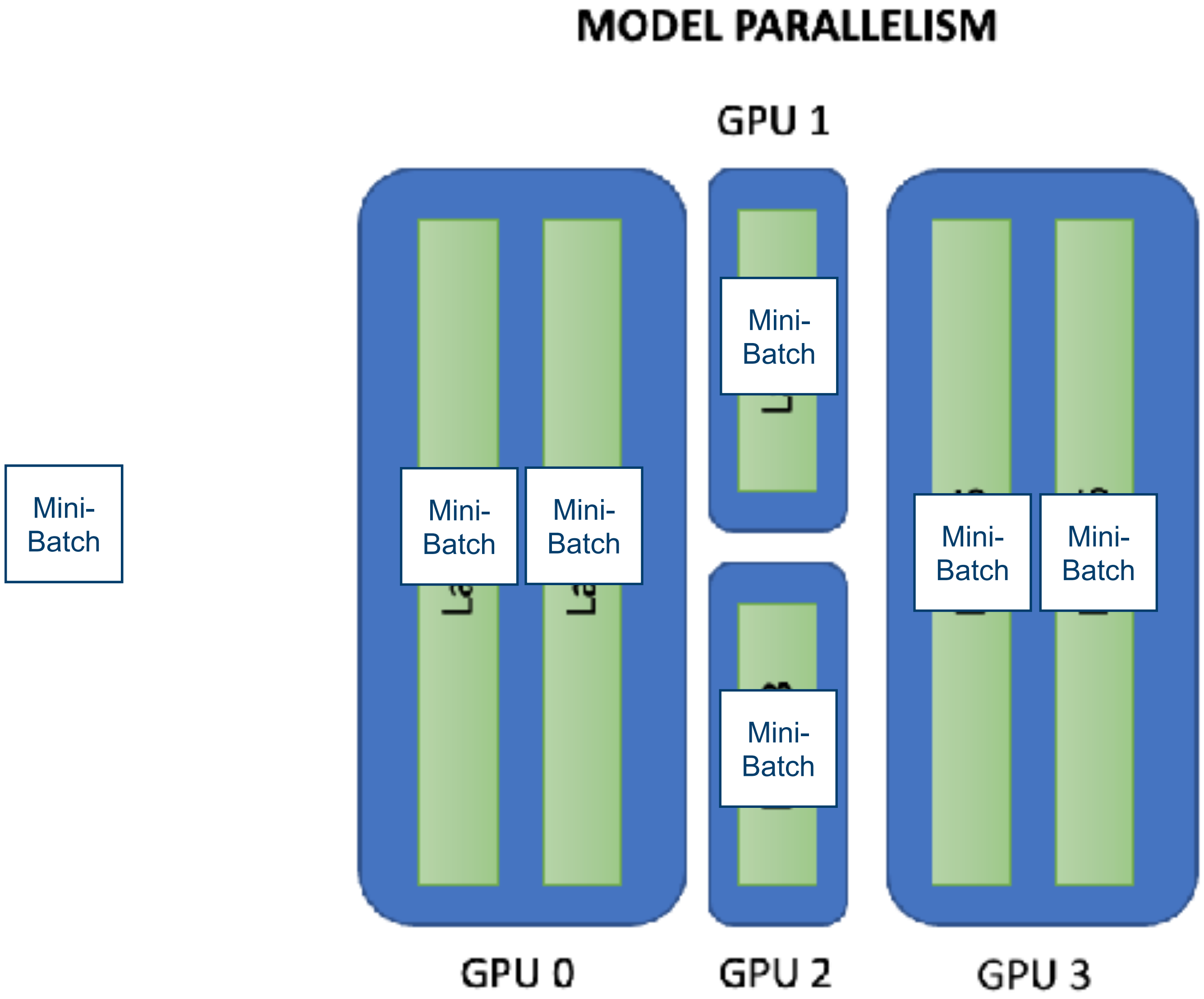
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

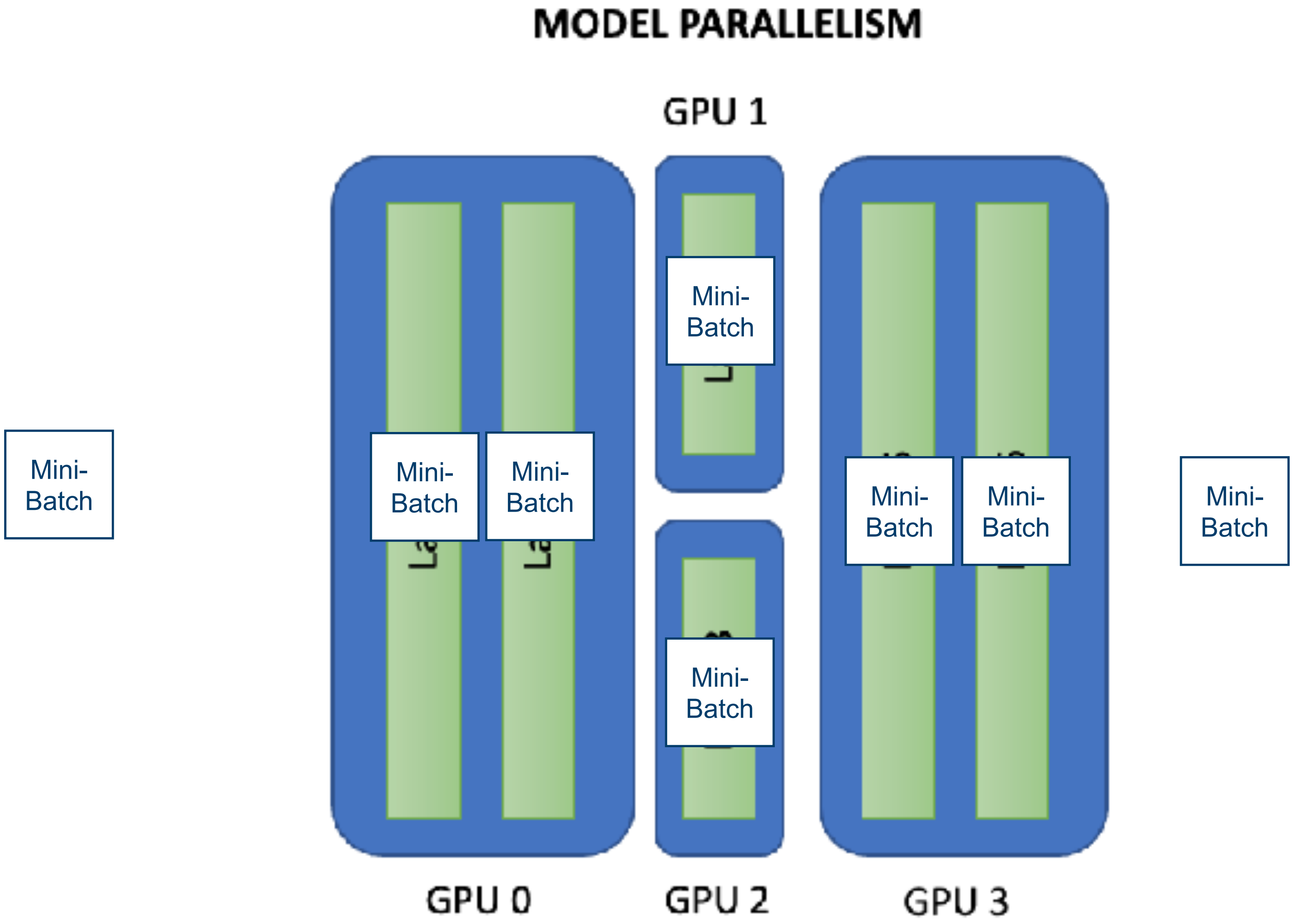
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

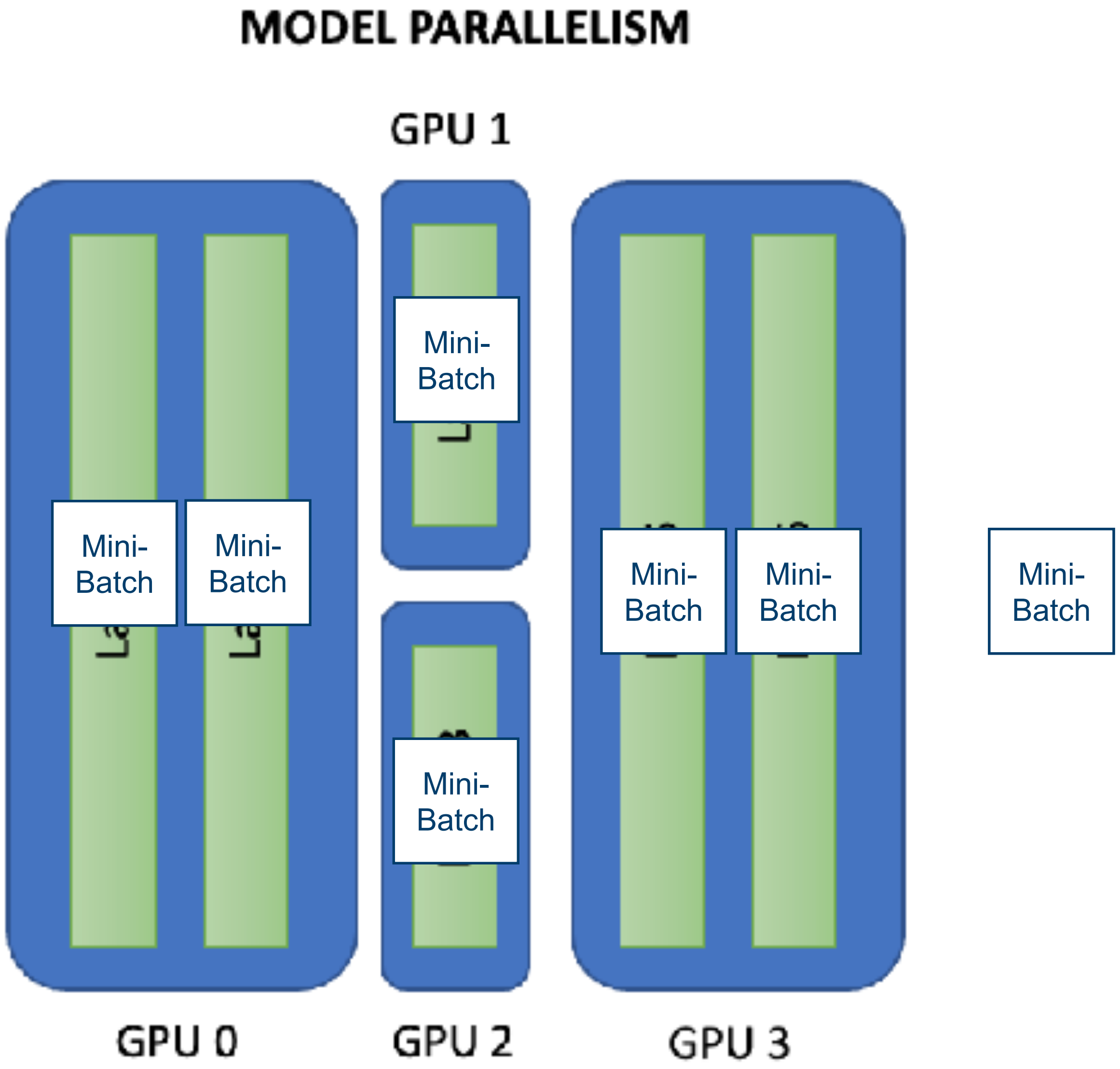
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Model Parallel



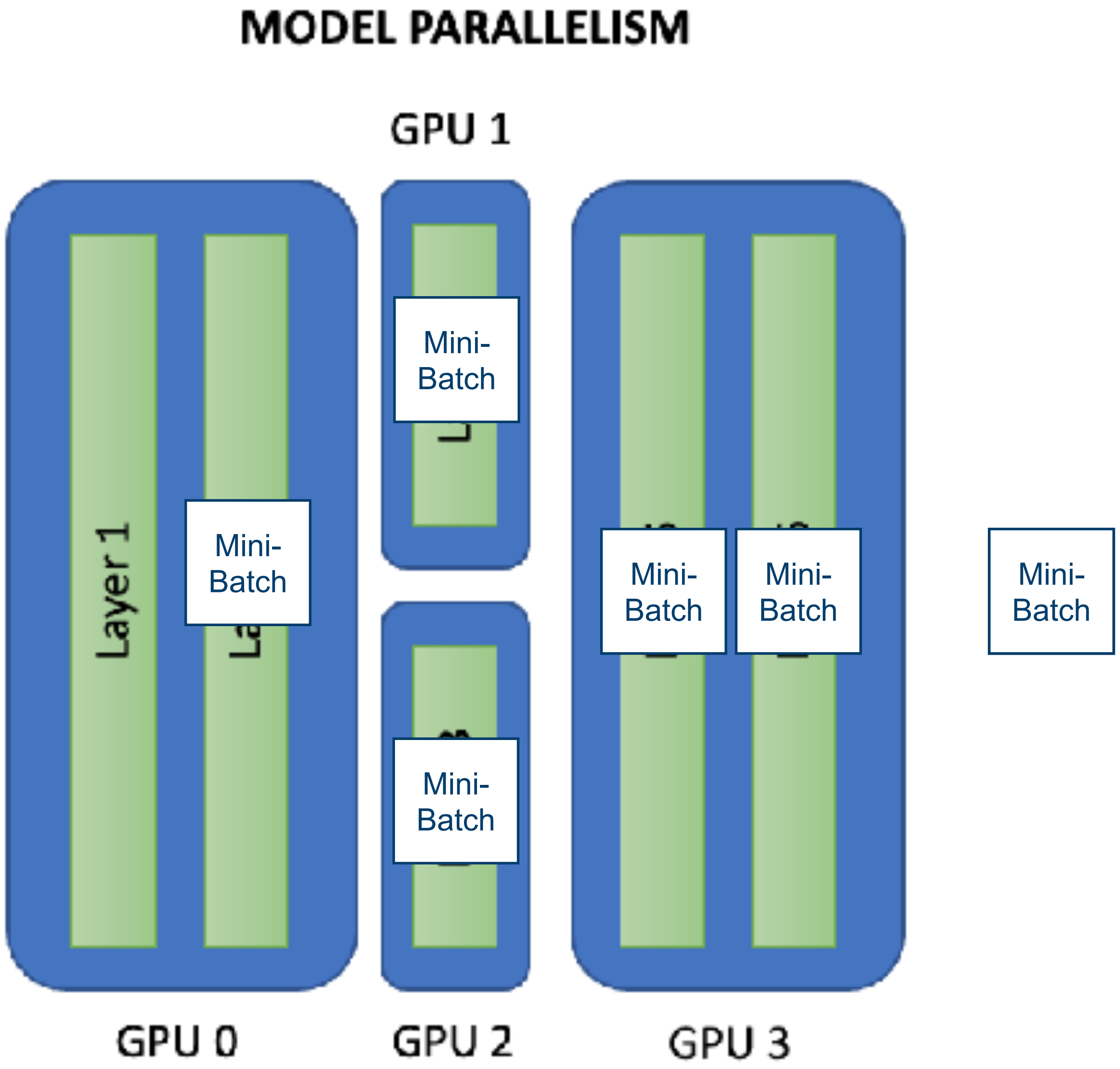

Jordi Torres.AI: Deep Learning on Supercomputers



JÜLICH Forschungszentrum

# Distributed training

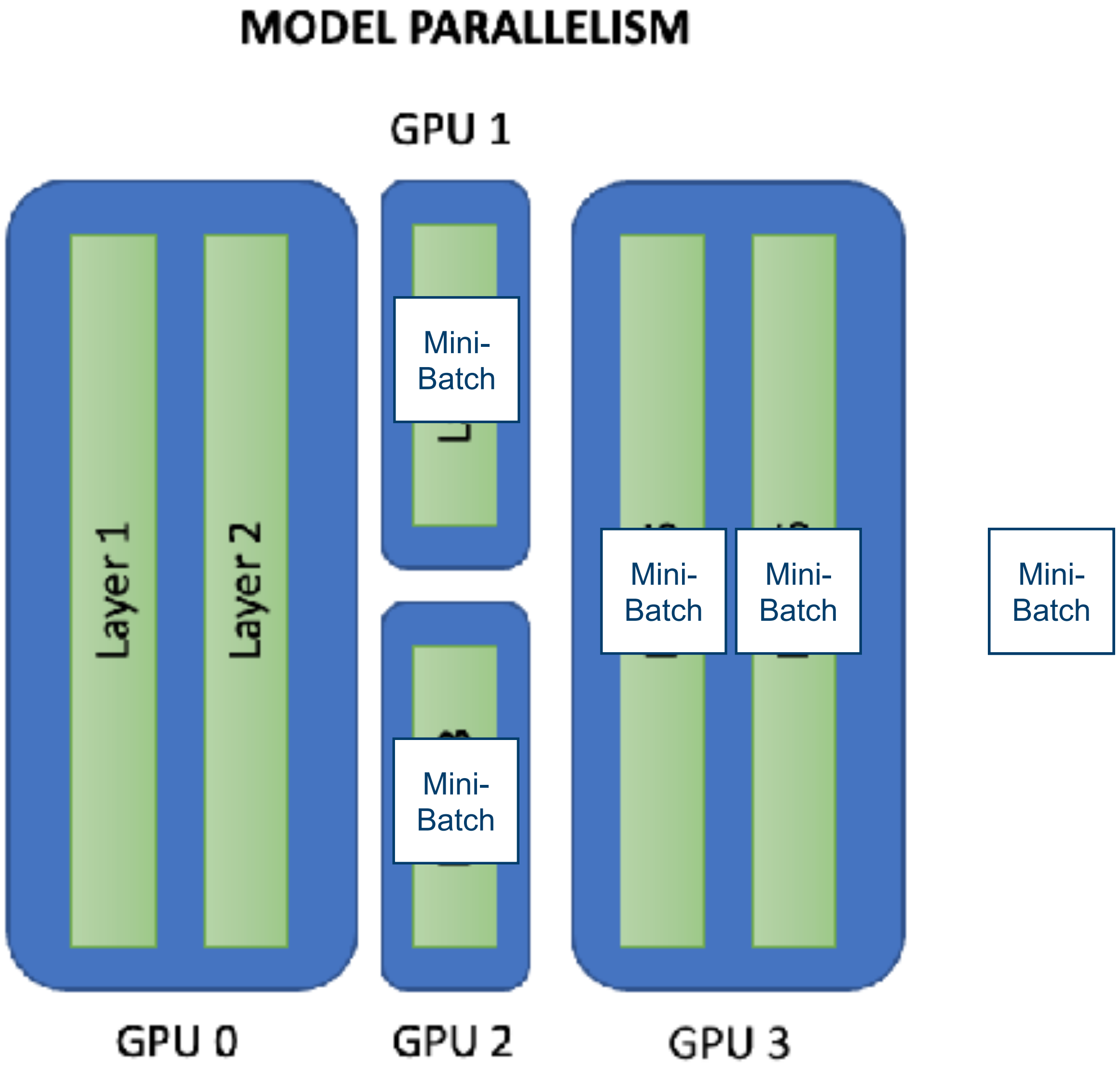
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

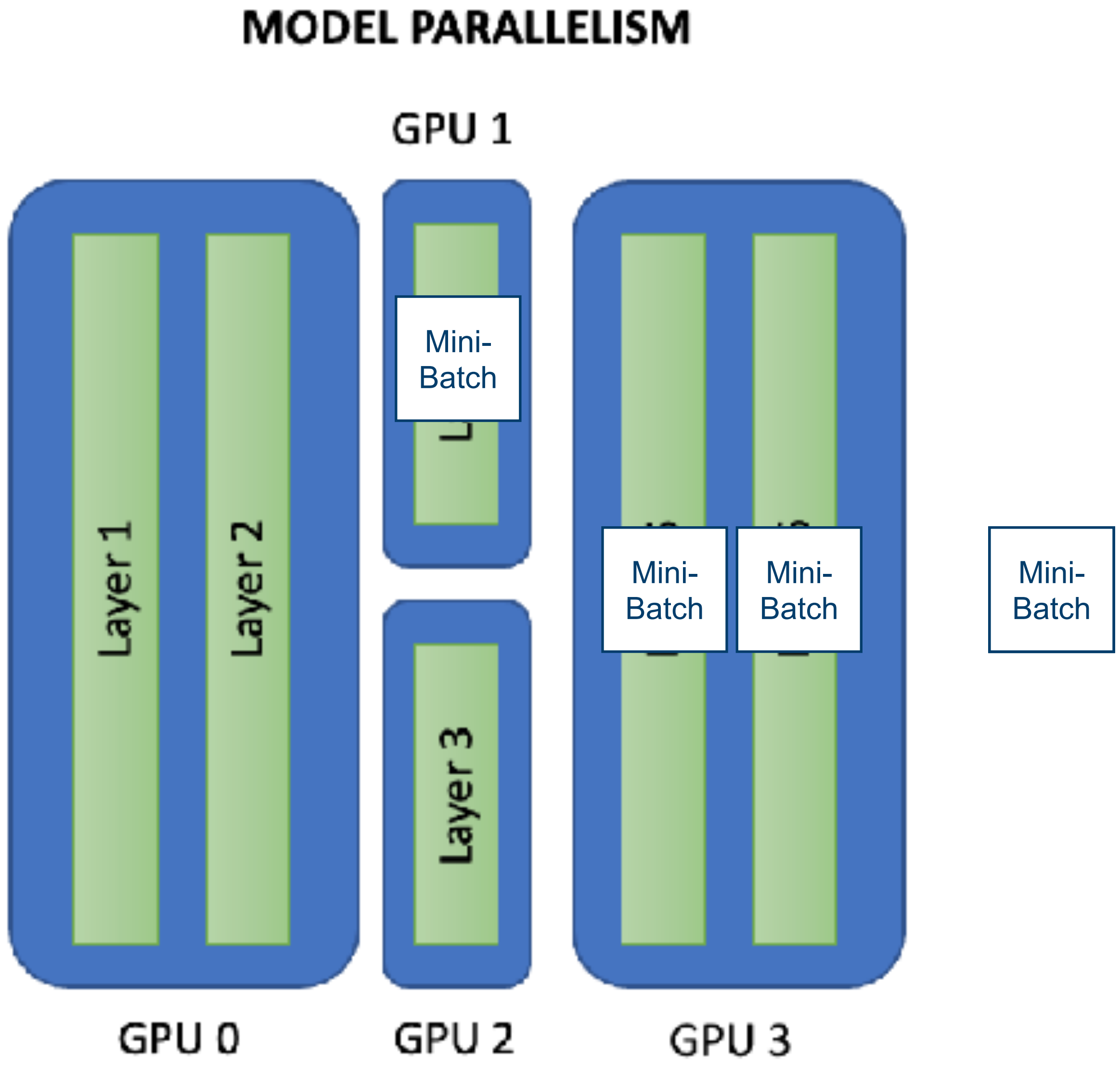
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

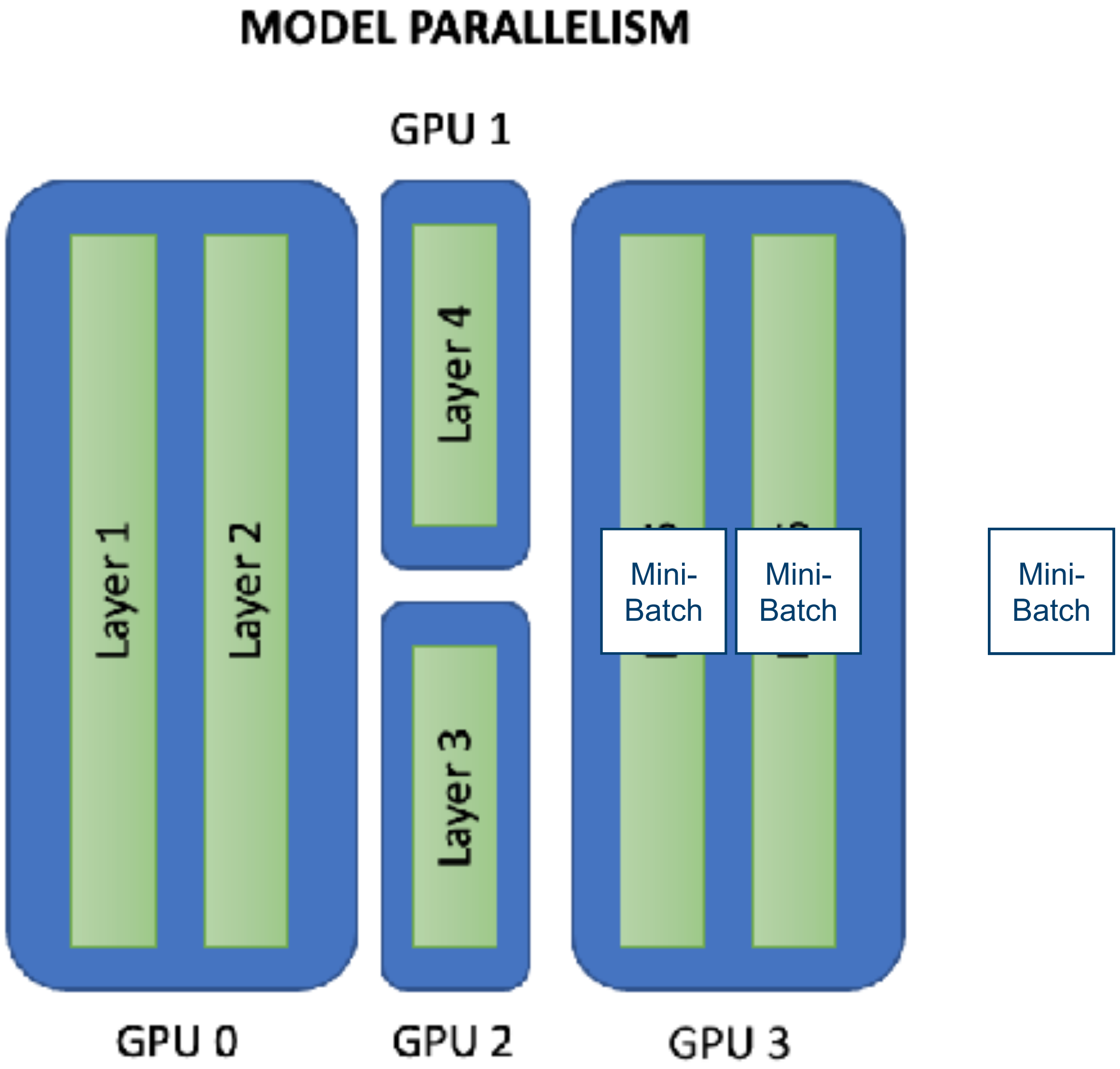
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Model Parallel



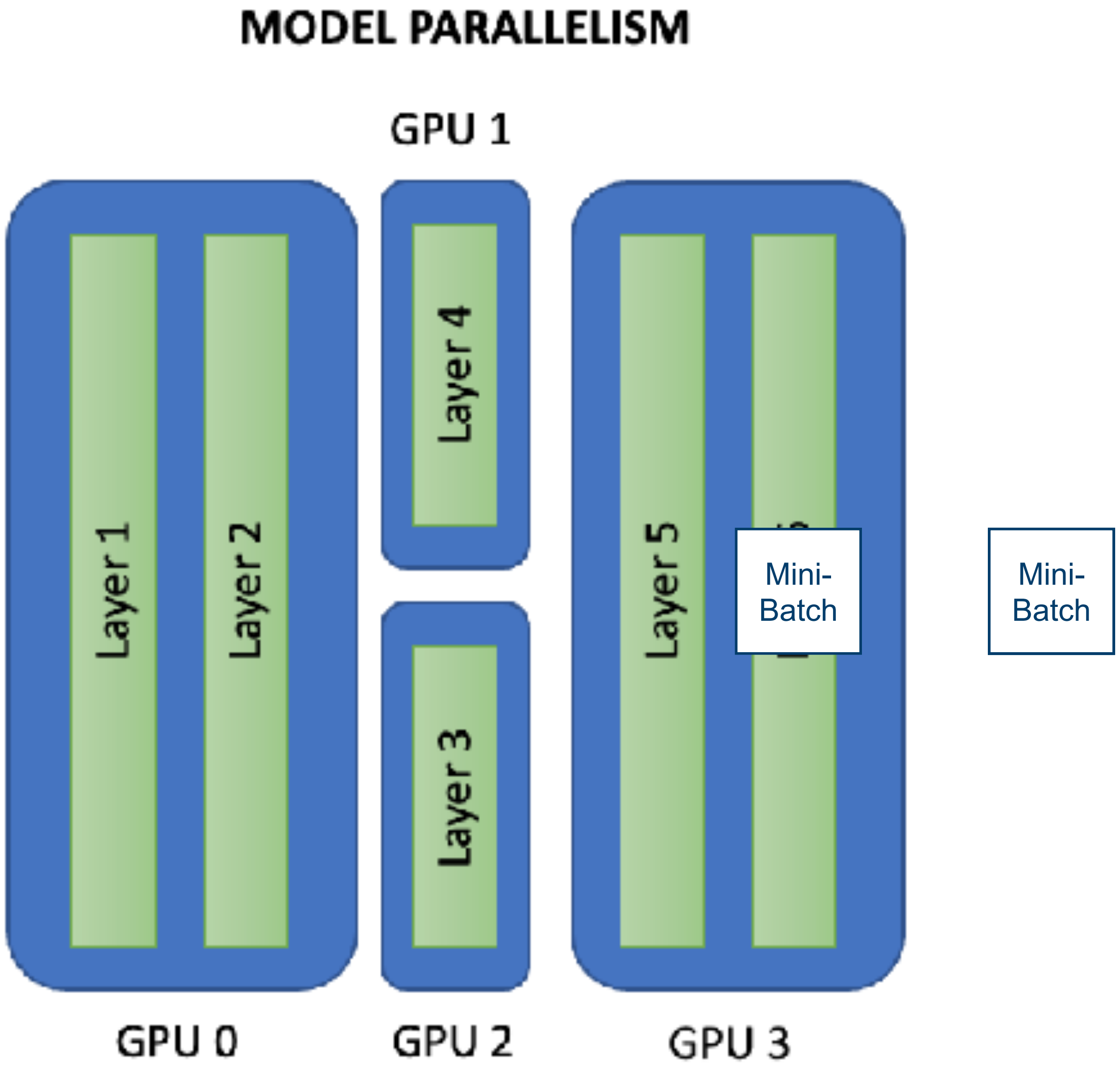

Jordi Torres.AI: Deep Learning on Supercomputers



JÜLICH Forschungszentrum

# Distributed training

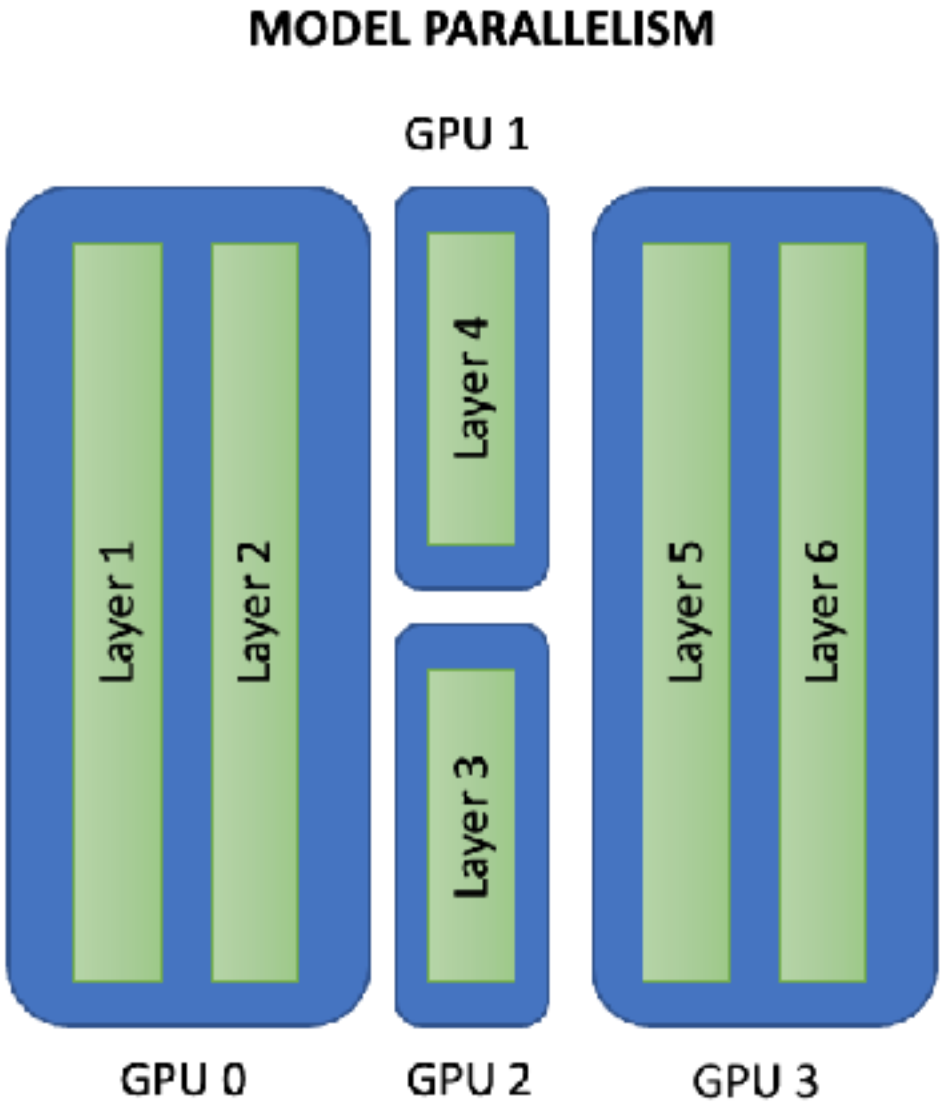
## Model Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Model Parallel




Mini-
Batch

Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Gradient averaging in data parallel training

Initialization $G_0 \boxed{\begin{matrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{matrix}}$ $G_1 \boxed{\begin{matrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{matrix}}$ $\cdots$ $G_{n-1} \boxed{\begin{matrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{matrix}}$



JÜLICH Forschungszentrum

# Distributed training

## Gradient averaging in data parallel training

Initialization

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

---

$$\vdots \qquad \vdots \qquad \cdots \qquad \vdots$$

Iteration 1

$$G_0 \begin{bmatrix} 0.2 & 0.1 \\ 0.9 & 0.3 \end{bmatrix} \quad G_1 \begin{bmatrix} 0.3 & 0.9 \\ 0.1 & 0.1 \end{bmatrix} \quad \cdots \quad G_{n-1} \begin{bmatrix} 0.1 & 0.5 \\ 0.9 & 0.6 \end{bmatrix}$$

$$\vdots \qquad \vdots \qquad \cdots \qquad \vdots$$

# Distributed training

## Gradient averaging in data parallel training

Initialization (Random values)

$G_0 \begin{bmatrix} 0.1 & 0.2 \\ 0.8 & 0.4 \end{bmatrix}$ $G_1 \begin{bmatrix} 0.4 & 0.2 \\ 0.8 & 0.2 \end{bmatrix}$ $\cdots$ $G_{n-1} \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix}$

$\vdots$ $\vdots$ $\cdots$ $\vdots$

Iteration 1

$G_0 \begin{bmatrix} 0.2 & 0.1 \\ 0.9 & 0.3 \end{bmatrix}$ $G_1 \begin{bmatrix} 0.3 & 0.9 \\ 0.1 & 0.1 \end{bmatrix}$ $\cdots$ $G_{n-1} \begin{bmatrix} 0.1 & 0.5 \\ 0.9 & 0.6 \end{bmatrix}$

$\vdots$ $\vdots$ $\cdots$ $\vdots$

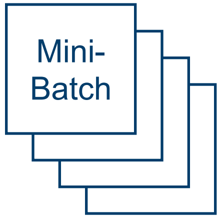
Average gradients across all GPUs

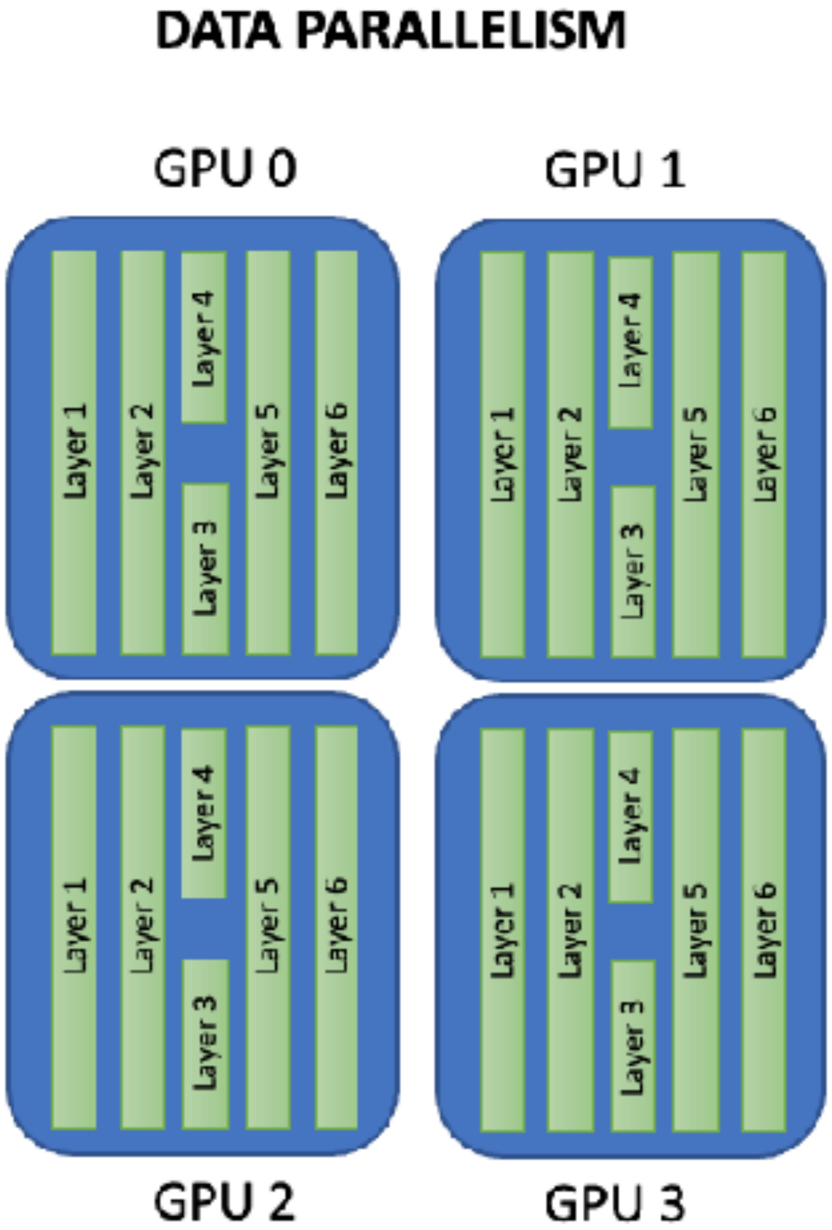
Updated Model

$G_0 \begin{bmatrix} 0.1 & 0.4 \\ 0.3 & 0.5 \end{bmatrix}$ $G_1 \begin{bmatrix} 0.1 & 0.4 \\ 0.3 & 0.5 \end{bmatrix}$ $\cdots$ $G_{n-1} \begin{bmatrix} 0.1 & 0.4 \\ 0.3 & 0.5 \end{bmatrix}$

# Distributed training

## Data Parallel

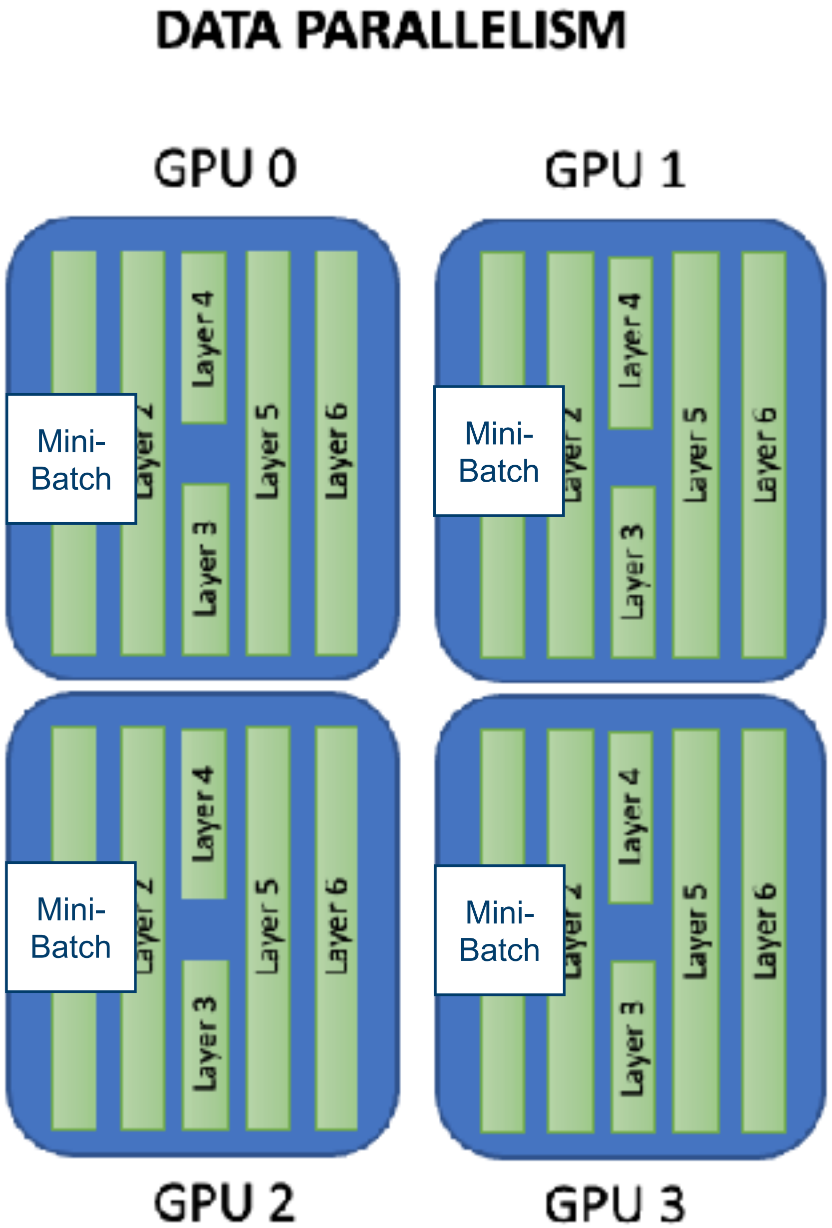



Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

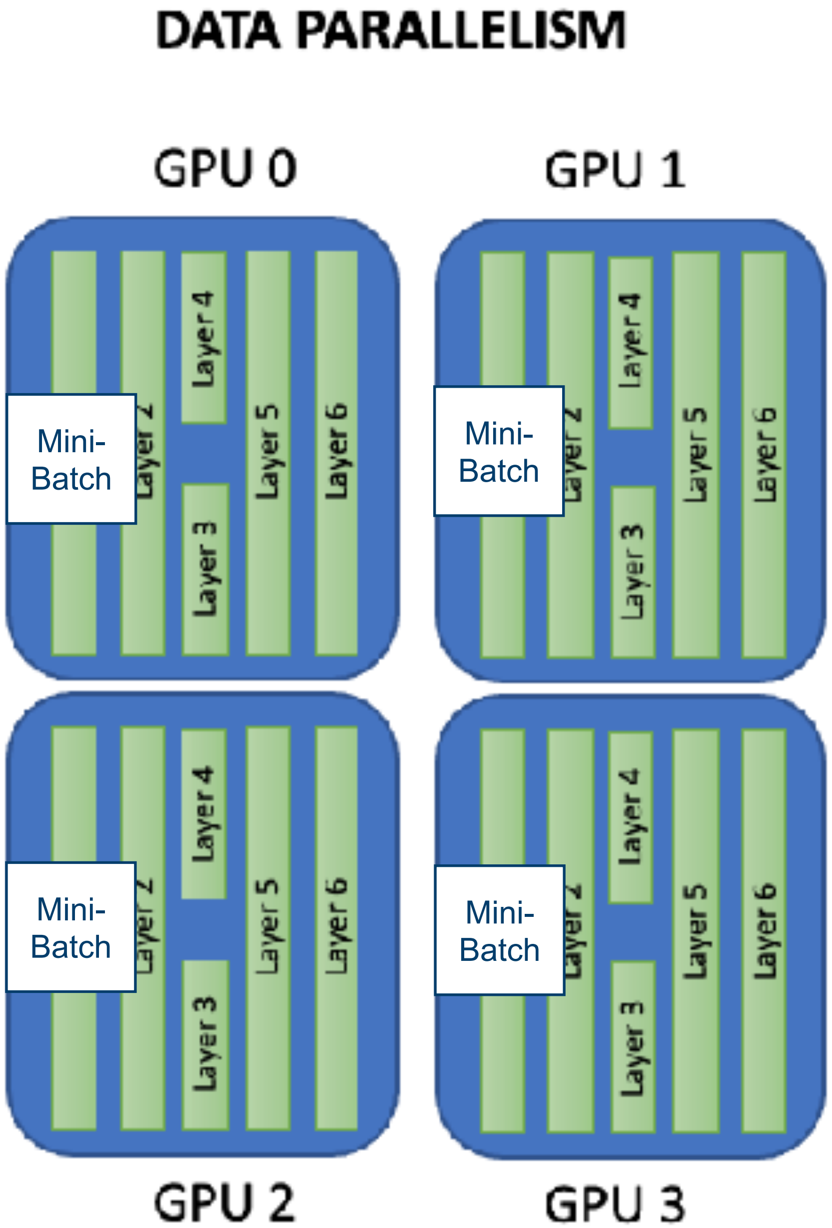
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

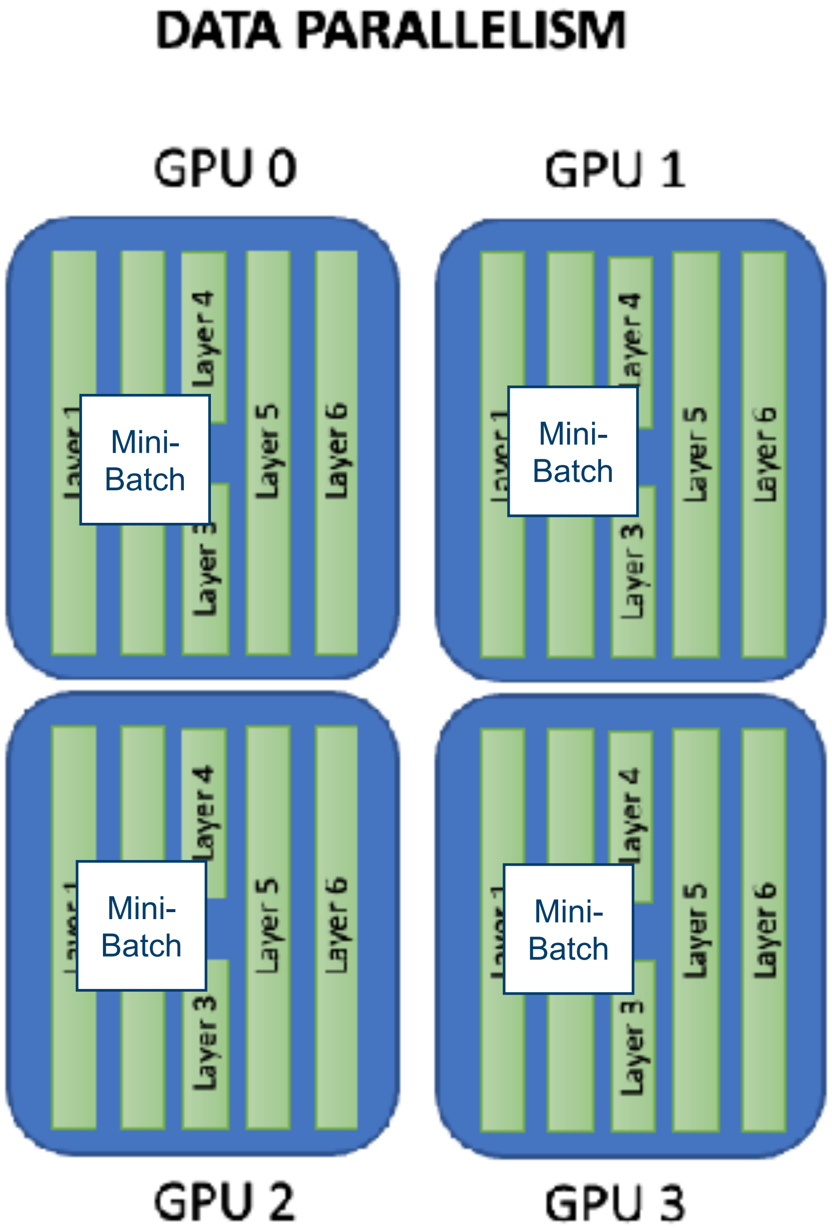
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

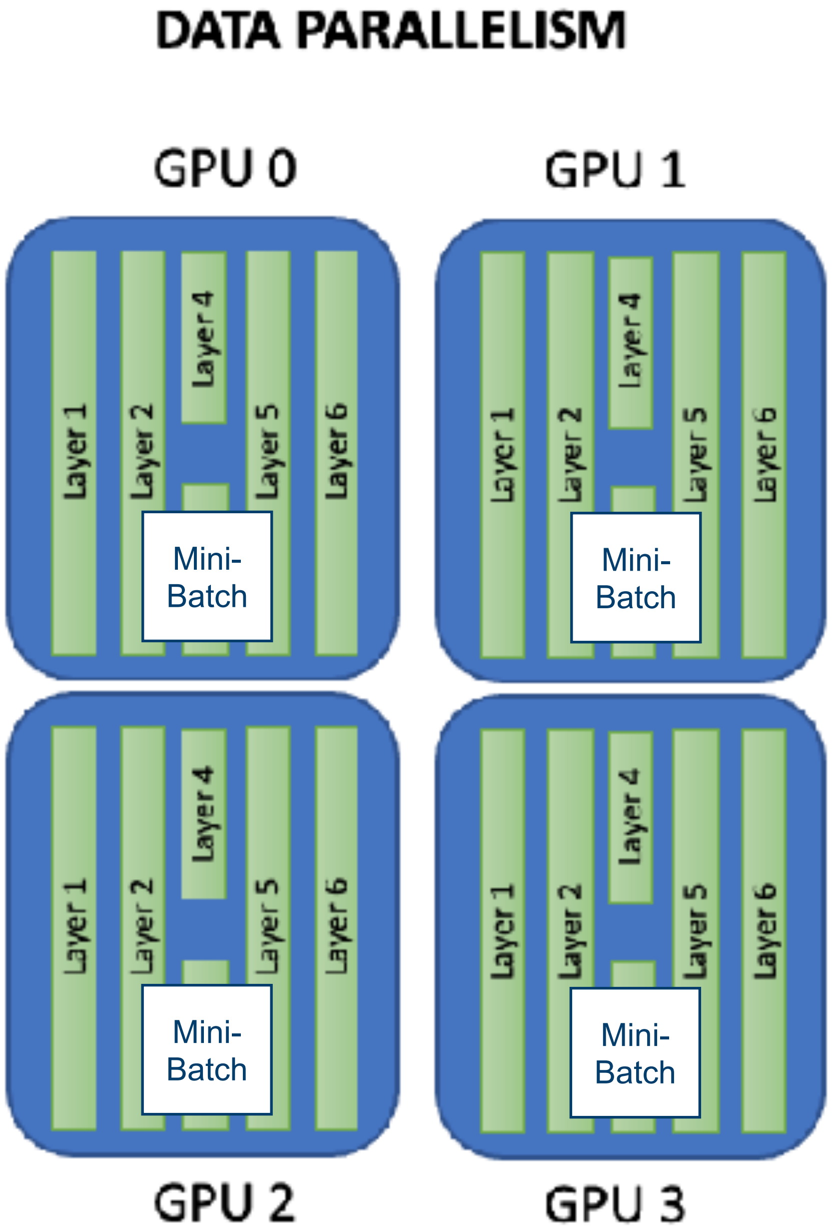
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

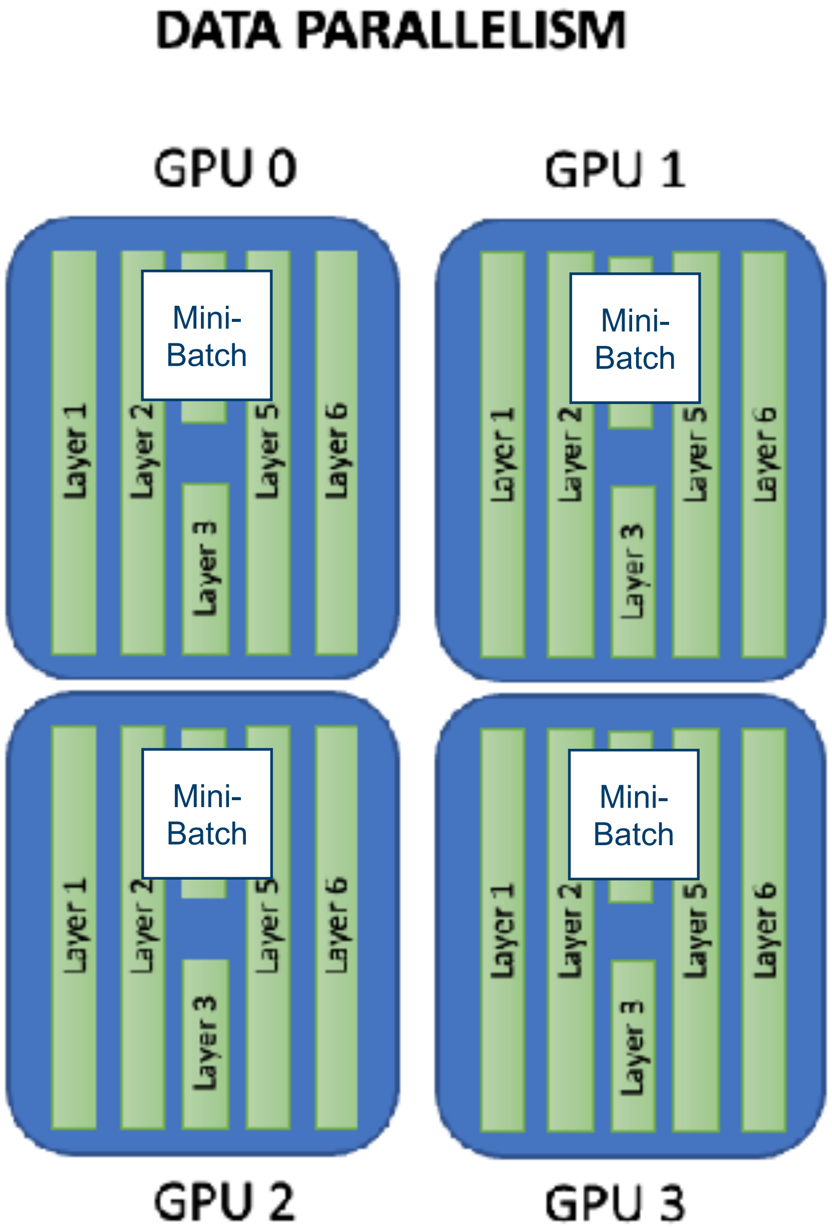
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

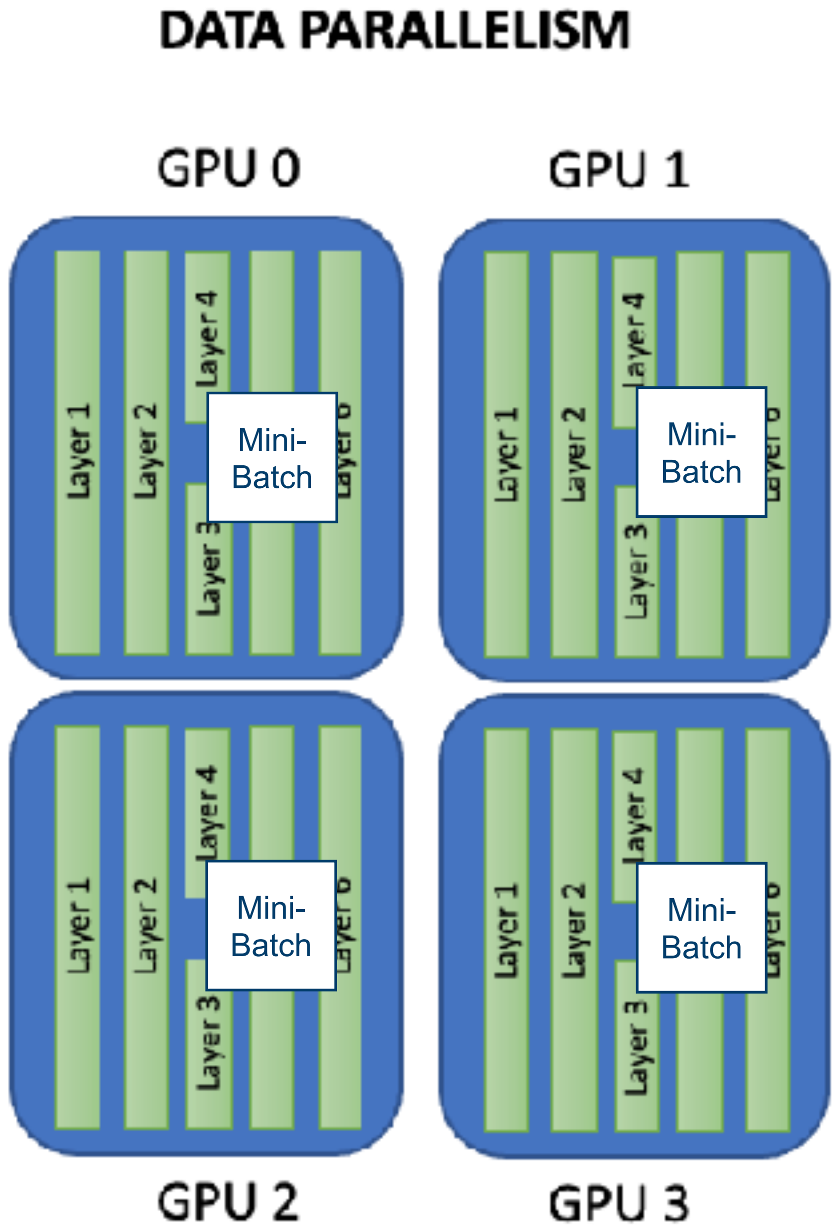
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

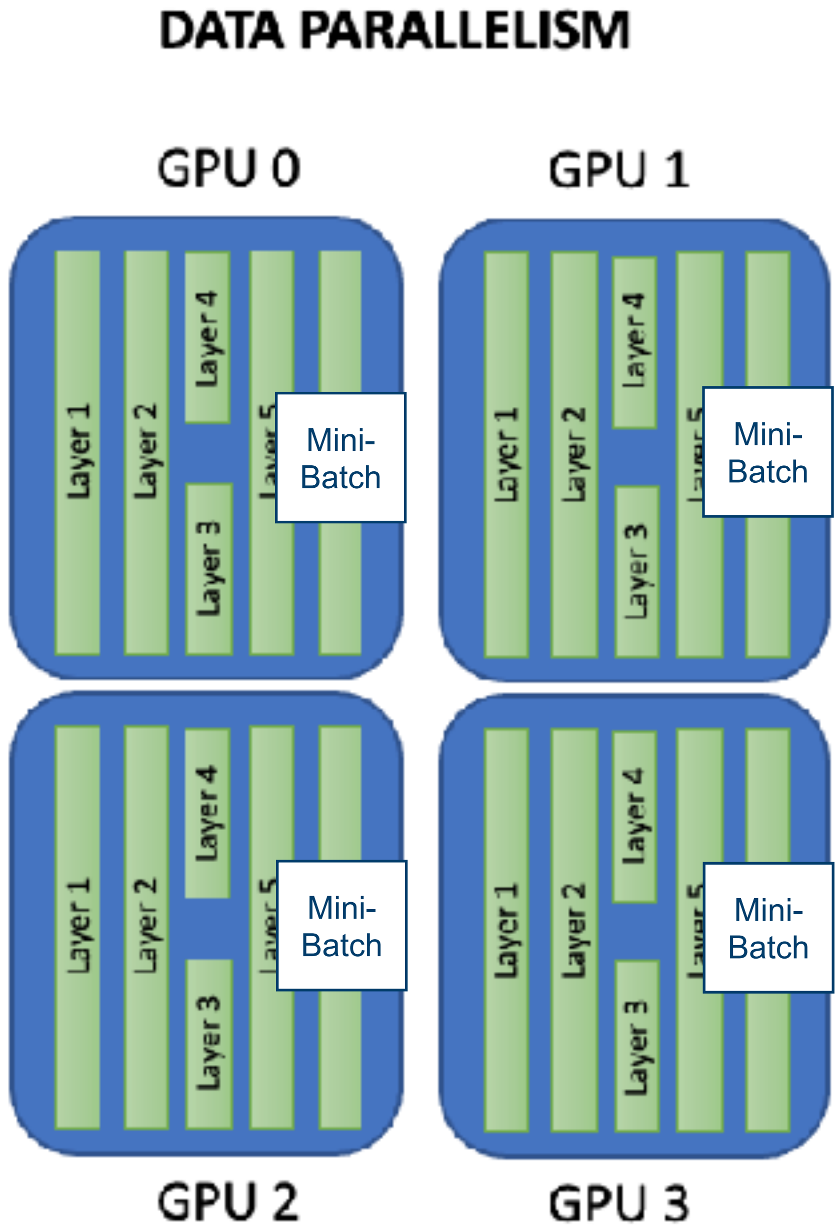
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

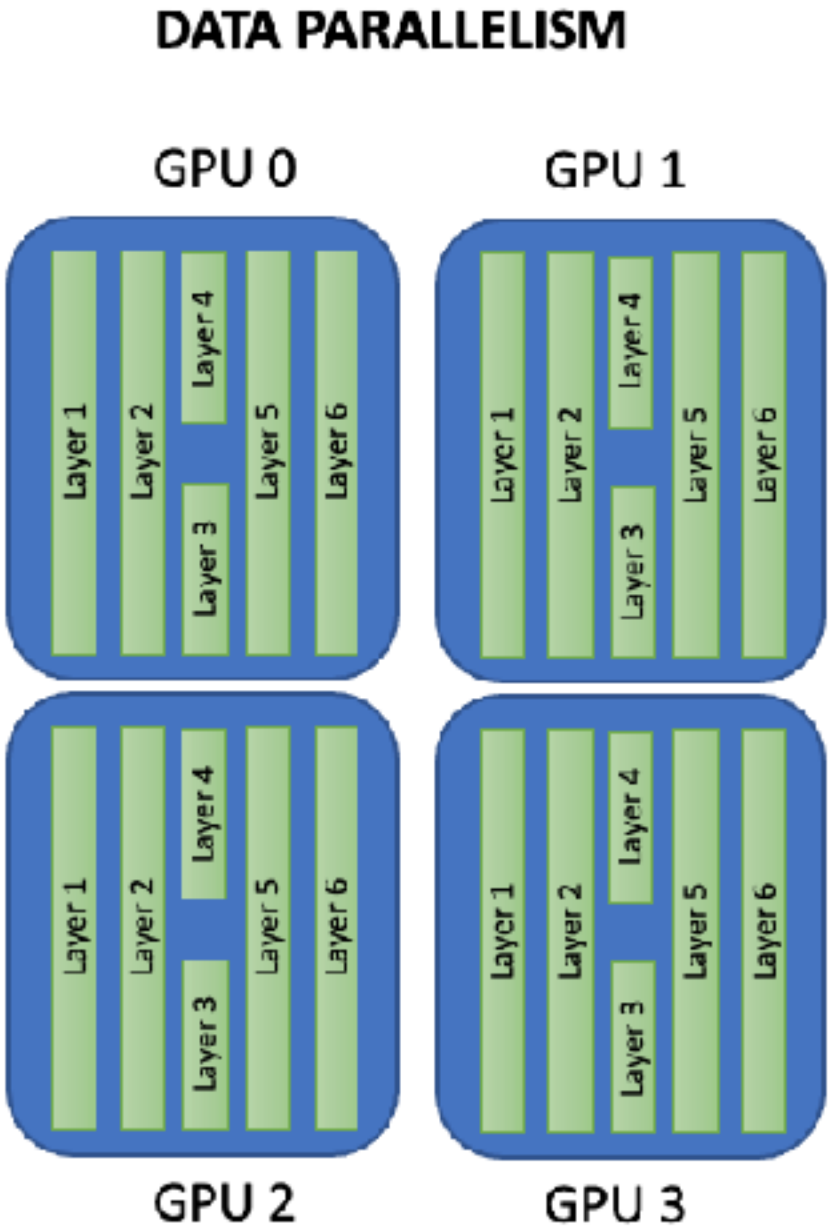
## Data Parallel




Jordi Torres.AI: Deep Learning on Supercomputers

# Distributed training

## Data Parallel




Mini-Batch

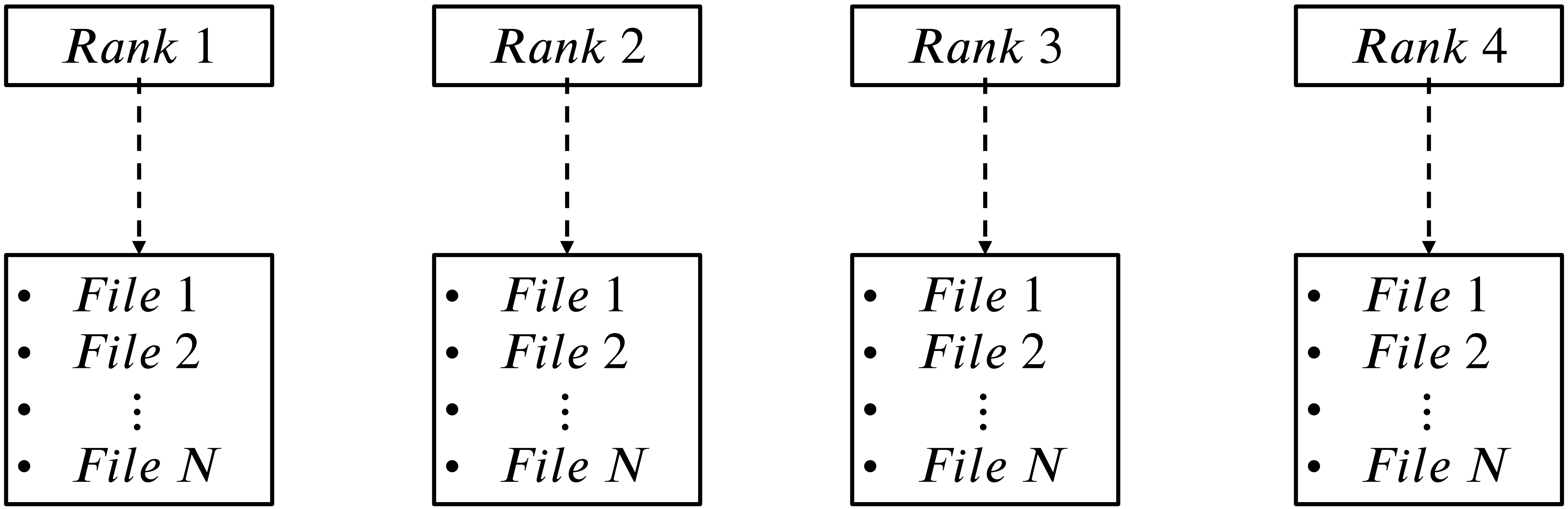
Jordi Torres.AI: Deep Learning on Supercomputers

# 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. MNIST classification: Epoch distributed
   ii. 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

## 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)

Single GPU

```
import tensorflow as tf
```

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
```

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
import math
```

dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
import math
import horovod.tensorflow.keras as hvd
```

dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
import math
import horovod.tensorflow.keras as hvd
hvd.init()
```

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
import math
import horovod.tensorflow.keras as hvd
hvd.init()
gpus = tf.config.experimental.list_physical_devices('GPU')
```

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```python
import tensorflow as tf
```

Distributed

```python
import tensorflow as tf
import math
import horovod.tensorflow.keras as hvd
hvd.init()
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()],
'GPU')
```

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`

# Distributed training: Examples

## Distributed training with Horovod: Code (1)

Single GPU

```
import tensorflow as tf
```

Distributed

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

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

```
mnist = tf.keras.datasets.mnist
```



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

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

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
```

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

<u>Single GPU</u>

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

<u>Distributed</u>

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()
```

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
])
optimizer = tf.keras.optimizers.Adam()
optimizer = hvd.DistributedOptimizer(optimizer)
```

# Distributed training: Examples

## Distributed training with Horovod: Code (2)

Single GPU

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
optimizer = tf.keras.optimizers.Adam()

epochs = 4
```

Distributed

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
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)
 ])
optimizer = tf.keras.optimizers.Adam()
optimizer =
hvd.DistributedOptimizer(optimizer)
epochs = int(math.ceil(4.0 / hvd.size()))
```

# Distributed training: Examples

## Distributed training with Horovod: Code (3)

<u>Single GPU</u>

```
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


      )
```

# Distributed training: Examples

## Distributed training with Horovod: Code (3)

Single GPU

```
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

     )
```

Distributed

```
14.  model.compile(
         optimizer=optimizer,
         loss='sparse_categorical_crossentropy',
         metrics=['accuracy']
      )
```

# Distributed training: Examples

## Distributed training with Horovod: Code (3)

Single GPU

```
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

    )
```

Distributed

```
14. model.compile(
        optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
     )
15. callbacks =
    [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
```

# Distributed training: Examples

## Distributed training with Horovod: Code (3)

Single GPU

```
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
    )
```

Distributed

```
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
     )
```



JÜLICH Forschungszentrum

# Distributed training: Examples

## Distributed training with Horovod: Output

# Distributed training: Examples

## Distributed training with Horovod: Output

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

# Distributed training: Examples

## Distributed training with Horovod: Output

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

Epoch 1/1
```



JÜLICH Forschungszentrum

# 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
```

# 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
```



JÜLICH Forschungszentrum

# Distributed training: Examples

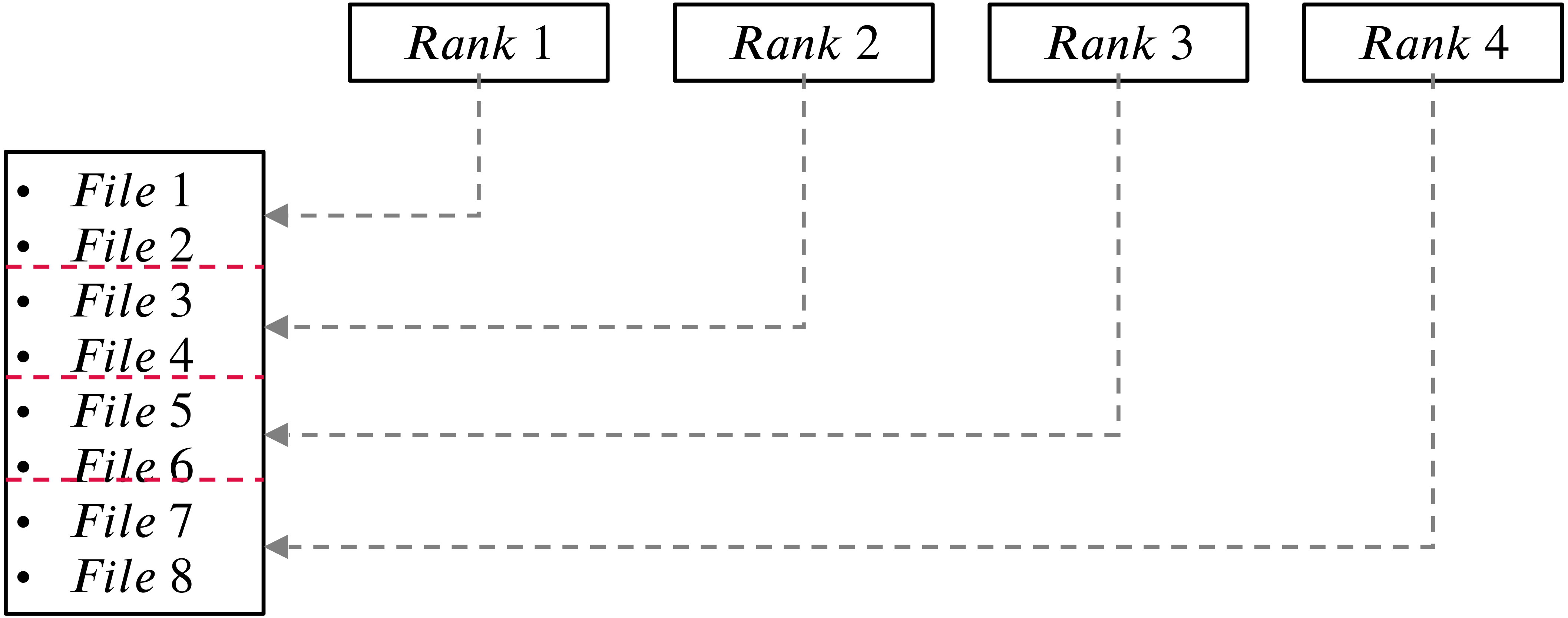
## Distributed training with Horovod: Output

```
$ 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: 0.07619360133947343
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

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) = 
                  mnist.load_data()
x_train = x_train / 255.0
...
```

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf
```

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) =
                    mnist.load_data()
x_train = x_train / 255.0
...
```


JÜLICH Forschungszentrum

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                 list(
                      islice(set, rank, None, world_size)
                      )
            ,float)
```

```
mnist = tf.keras.datasets.mnist
       (x_train, y_train), (x_test, y_test) =
                 mnist.load_data()
x_train = x_train / 255.0
...
```

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                 list(
                      islice(set, rank, None, world_size) [Iterable]
                      )
           ,float)
```

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) =
                       mnist.load_data()
x_train = x_train / 255.0
...
```

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                list(
                    islice(set, rank, None, world_size)
                    )
            ,float)
```

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) =
                  mnist.load_data()
x_train = x_train / 255.0
...
```

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                list(
                     islice(set, rank, None, world_size)
                     )
           ,float)
```

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) =
                      mnist.load_data()
x_train = x_train / 255.0
...
```



JÜLICH Forschungszentrum

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                    list(
                         islice(set, rank, None, world_size)
                         )
              ,float)

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




x_train_local = x_train_local / 255.0
```

```
mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) =
                  mnist.load_data()
x_train = x_train / 255.0
...
```

# Distributed training: Examples

## Input data distribution for Horovod: Code snippet

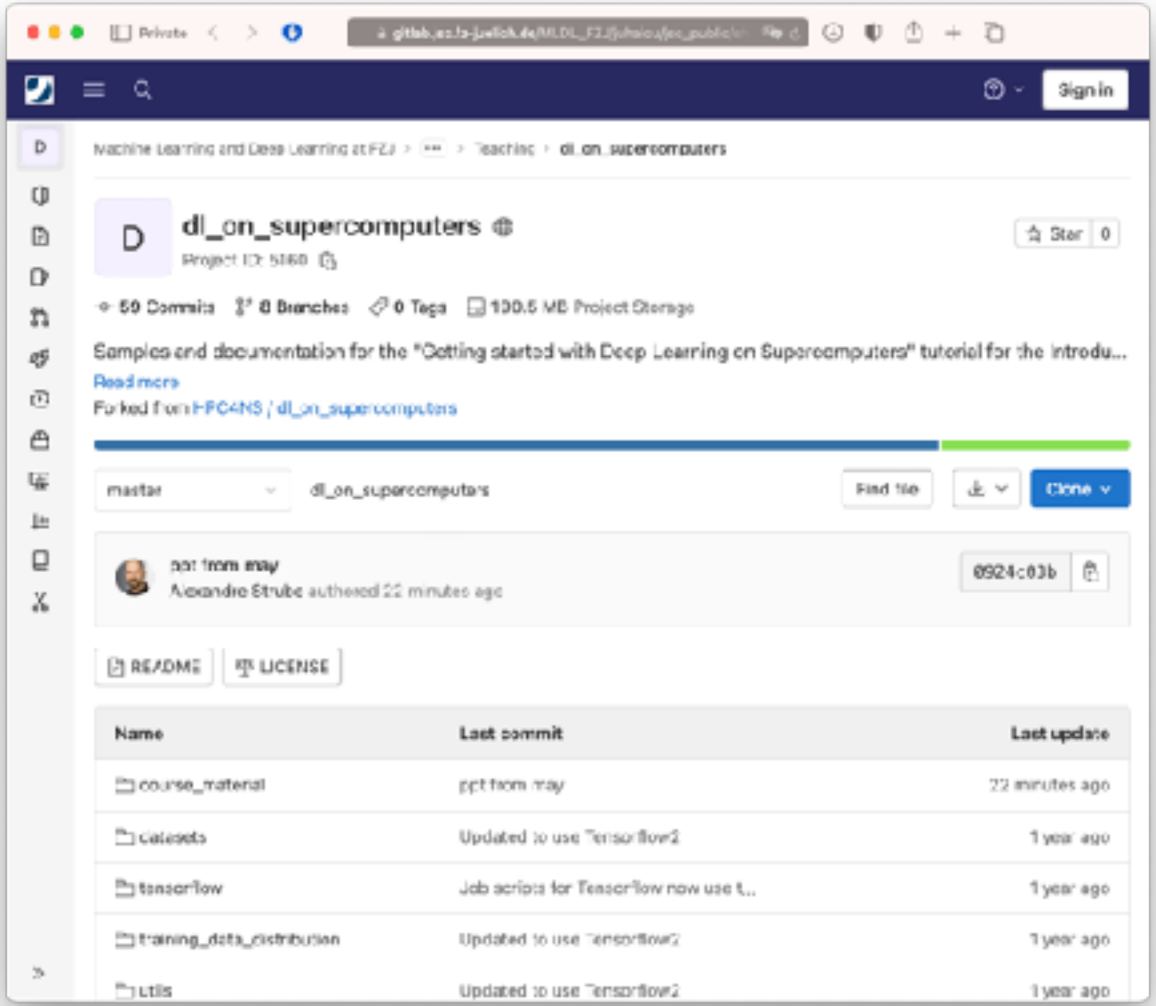
```
from mpi4py import MPI
from itertools import islice
import numpy
Import tensorflow as tf

def slice(set):
    return numpy.asarray(
                list(
                     islice(set, rank, None, world_size)
                     )
          ,float)

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train_local = slice(x_train)
y_train_local = slice(y_train)
x_test_local  = slice(x_test)
y_test_local  = slice(y_test)
x_train_local = x_train_local / 255.0
```

```
mnist = tf.keras.datasets.mnist
       (x_train, y_train), (x_test, y_test) =
                 mnist.load_data()
x_train = x_train / 255.0
...
```

# Agenda

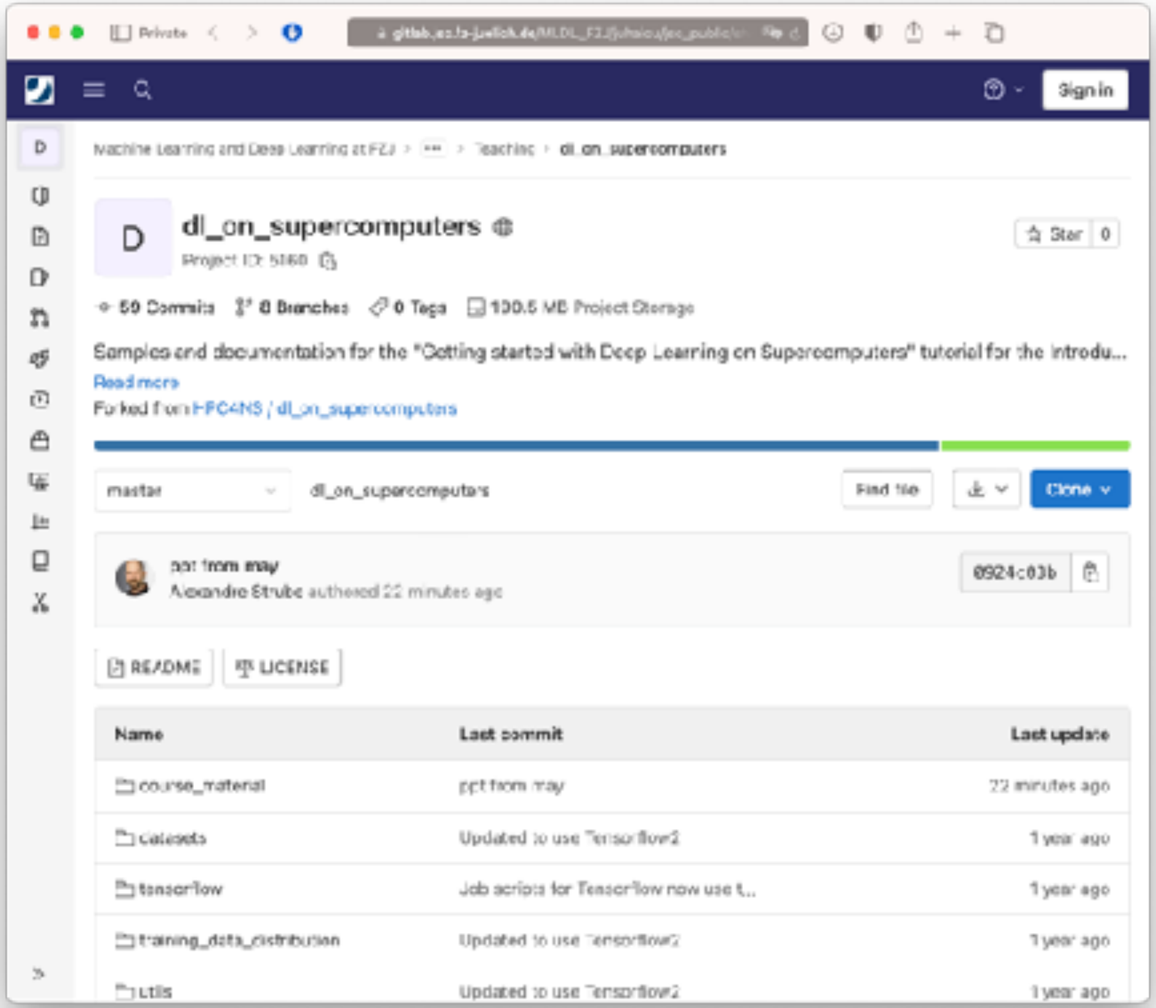
# Deep learning on Supercomputers

## The tutorial



# Deep learning on Supercomputers

## The tutorial




https://go.fzj.de/ml_on_supercomputers

# 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

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.er
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.er
#SBATCH --time=00:10:00
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.er
#SBATCH --time=00:10:00
#SBATCH --job-name=TUTORIAL
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.er
#SBATCH --time=00:10:00
#SBATCH --job-name=TUTORIAL
#SBATCH --gres=gpu:4 --partition=dc-gpu-devel --account <…>
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
```



JÜLICH Forschungszentrum

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
```



JÜLICH Forschungszentrum

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

# Make all GPUs visible per node
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.er
#SBATCH --time=00:10:00
#SBATCH --job-name=TUTORIAL
#SBATCH --gres=gpu:4 --partition=dc-gpu-devel --account <…>
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<…>
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

# Make all GPUs visible per node
export CUDA_VISIBLE_DEVICES=0,1,2,3
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

# Make all GPUs visible per node
export CUDA_VISIBLE_DEVICES=0,1,2,3

```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

# Make all GPUs visible per node
export CUDA_VISIBLE_DEVICES=0,1,2,3

# Run the program
```

# Deep learning on Supercomputers

## The tutorial: Job script `jureca_job.sh`

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

```
# Load the required modules
module load GCC
module load OpenMPI
module load TensorFlow
module load Horovod

# Make all GPUs visible per node
export CUDA_VISIBLE_DEVICES=0,1,2,3

# Run the program
srun python -u keras_mnist.py
```

# Deep learning on Supercomputers

## The tutorial: Starting and monitoring the training job



JÜLICH Forschungszentrum

# 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
```

# 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
```

# 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
```

# 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)
```

# 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
```



JÜLICH Forschungszentrum

# 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
...
```



JÜLICH Forschungszentrum

# 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
...
```

# 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
```

# 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
```



JÜLICH Forschungszentrum

# Agenda





JÜLICH Forschungszentrum

# 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
- PyTorch, Tensorflow and Horovod are already available on all supercomputers
- 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

### Useful resources

Tutorial - https://go.fzj.de/ml_on_supercomputers

# 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
- PyTorch, Tensorflow and Horovod are already available on all supercomputers
- 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

**Useful resources**

Tutorial - https://go.fzj.de/ml_on_supercomputers

Thank you!