IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

APPLIED ACCELERATED ARTIFICIAL INTELLIGENCE

Accelerated TensorFlow

Dr. Satyajit Das
Assistant Professor
Data Science
Computer Science and Engineering
IIT Palakkad

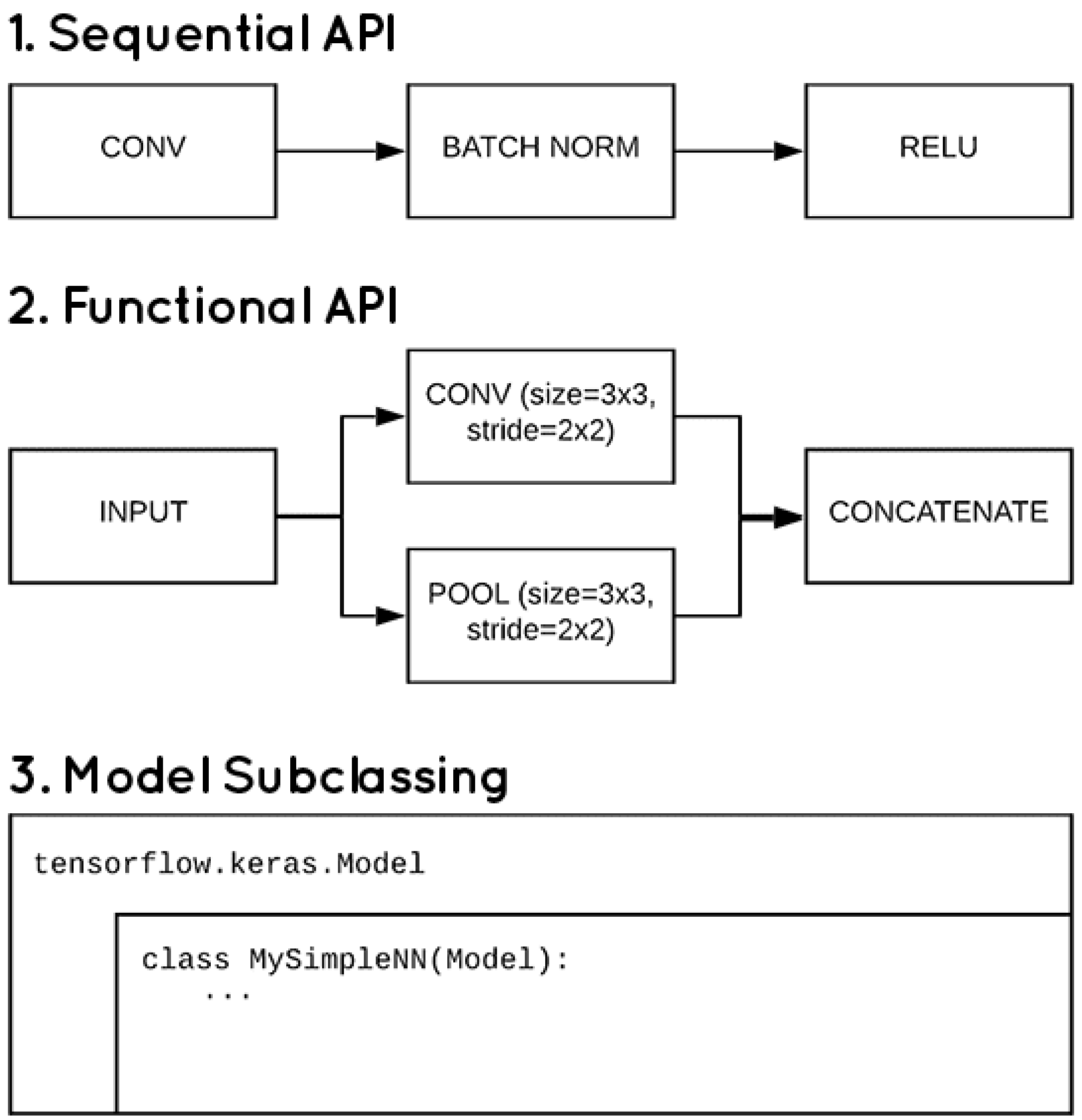
National Supercomputing Mission

Centre for Development of Advanced Computing

Topics

- 3 ways to create a model
- Accelerated data pipelining
- Distributed training

3 ways to create model



Quick overview

```
model = Sequential()

model.add(Dense(4,activation='relu')) ##<----- You don't have to specify input size.Just define the hidden layers
model.add(Dense(4,activation='relu’))
model.add(Dense(1))## defining the optimiser and loss function


model.compile(optimizer='adam',loss='mse')## training the model
model.fit(x=X_train,y=y_train, validation_data=(X_test,y_test), batch_size=128,epochs=400)
```

```
class WideAndDeepModel(keras.Model):
        def __init__(self, units=30, activation="relu", **kwargs):
        super().__init__(**kwargs) # handles standard aruments ( name, etc.)
       self.hidden1 = keras.layers.Dense(units, activation=activation)
       self.hidden2 = keras.layers.Dense(units, activation=activation)
       self.main_output = keras.layers.Dense(1)
       self.aux_output = keras.layers.Dense(1)

    def call(self, inputs):
        input_A, input_B = inputs
      hidden1 = self.hidden1(inputs_B)
      hidden2 = self.hidden2(hidden1)
      concat = keras.layers.Concatenate([input_A, hidden2])
      main_ouput = self.main_ouput(concat)
      aux_ouput = self.aux_ouput(hidden2)
      return main_ouput, aux_output

model = WideAndDeepModel()
```

```
## Creating the layers

input_layer = Input(shape=(3,))
Layer_1 = Dense(4, activation="relu")(input_layer)
Layer_2 = Dense(4, activation="relu")(Layer_1)
output_layer= Dense(1, activation="linear")(Layer_2)

##Defining the model by specifying the input and output layers

model = Model(inputs=input_layer, outputs=output_layer) ## defining the optimiser and loss function model.compile(optimizer='adam', loss='mse’)

## training the model model.fit(X_train, y_train,epochs=400, batch_size=128,validation_data=(X_test,y_test))
```

Why input pipeline is important

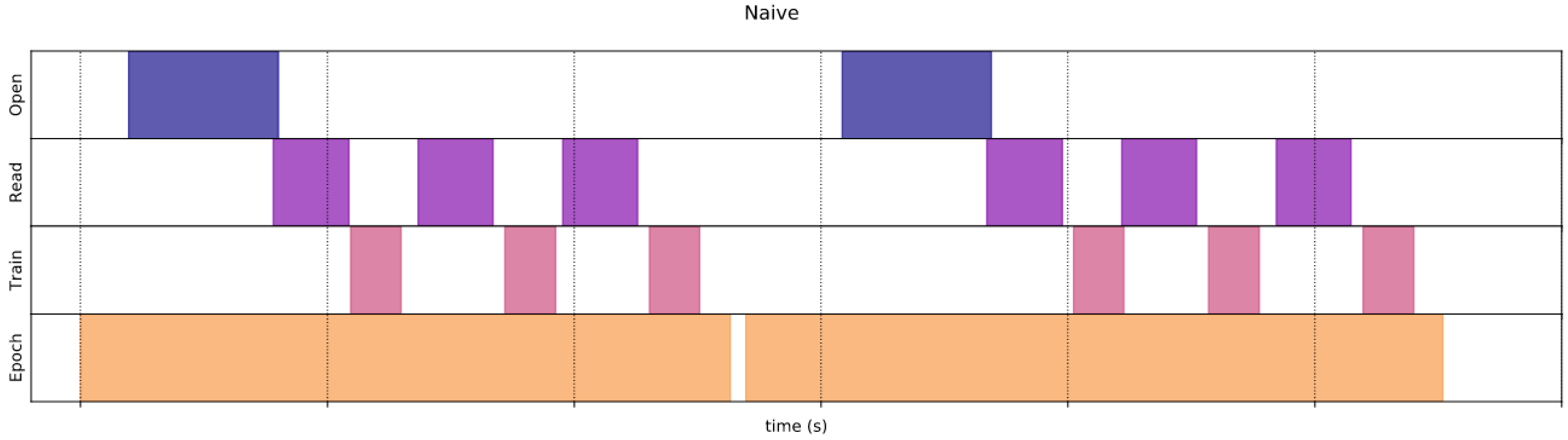
- data might not fit into memory
- data might require (randomized) pre-processing
- efficiently utilize hardware
- decouple loading + pre-processing from distribution

- Extract:
  - read data from memory / storage
  - parse file format
- Transform:
  - text vectorization
  - image transformations
  - video temporal sampling
  - shuffling, batching,...
- Load:
  - transfer data to the accelerator

The naive approach

```python
def benchmark(dataset, num_epochs=2):
    start_time = time.perf_counter()
    for epoch_num in range(num_epochs):
        for sample in dataset:
            # Performing a training step
            time.sleep(0.01)
    print("Execution time:", time.perf_counter() - start_time)
```

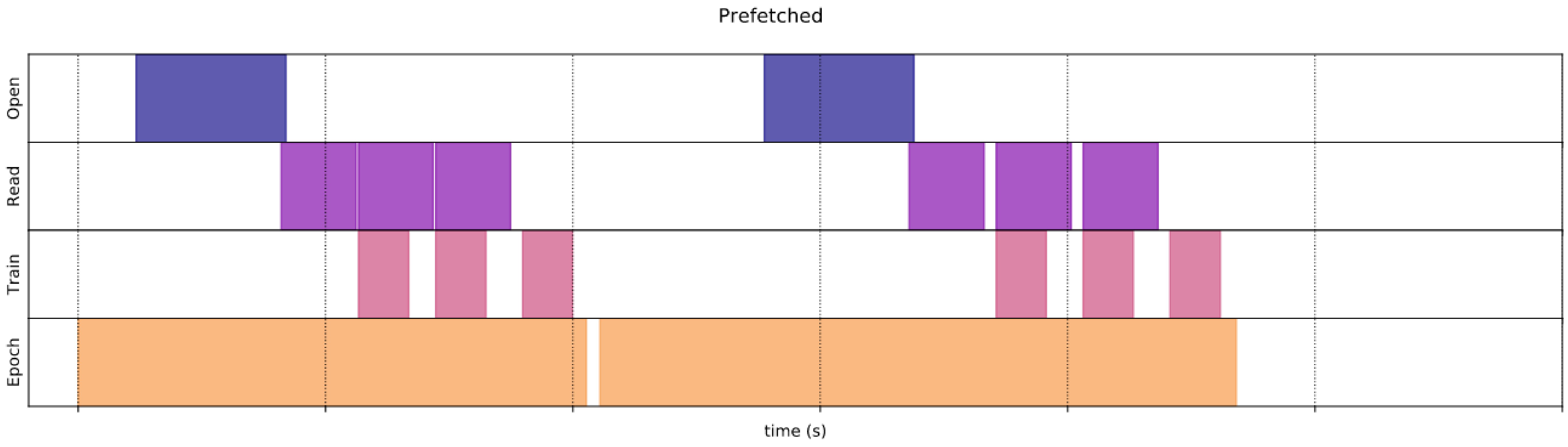
```python
benchmark(ArtificialDataset())
```



https://www.tensorflow.org/guide/data_performance

Prefetching

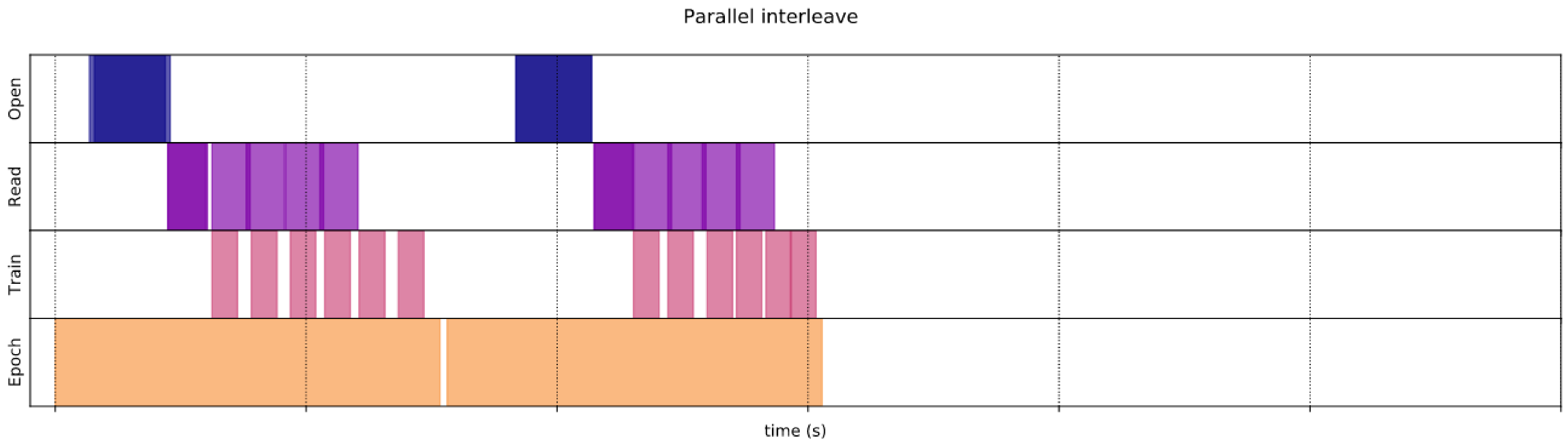
```
benchmark(
    ArtificialDataset()
    .prefetch(tf.data.AUTOTUNE)
)
```



https://www.tensorflow.org/guide/data_performance

Interleaving

```
benchmark(
    tf.data.Dataset.range(2)
    .interleave(
        lambda _: ArtificialDataset(),
        num_parallel_calls=tf.data.AUTOTUNE
    )
)
```

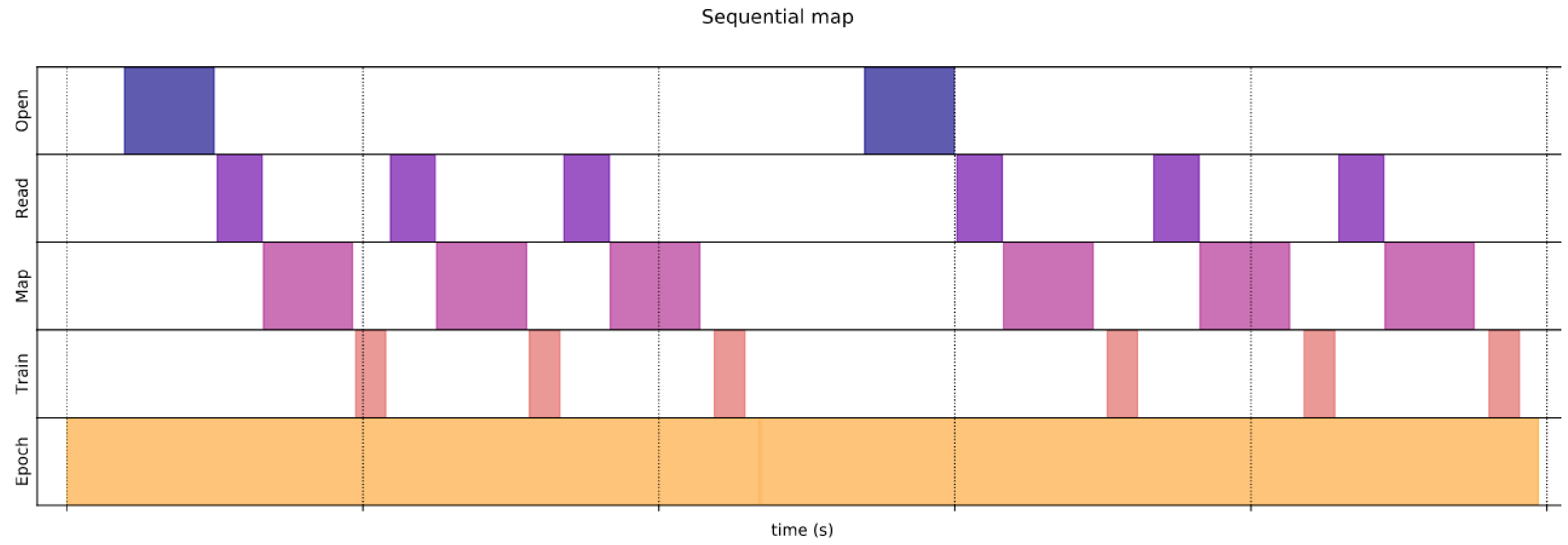


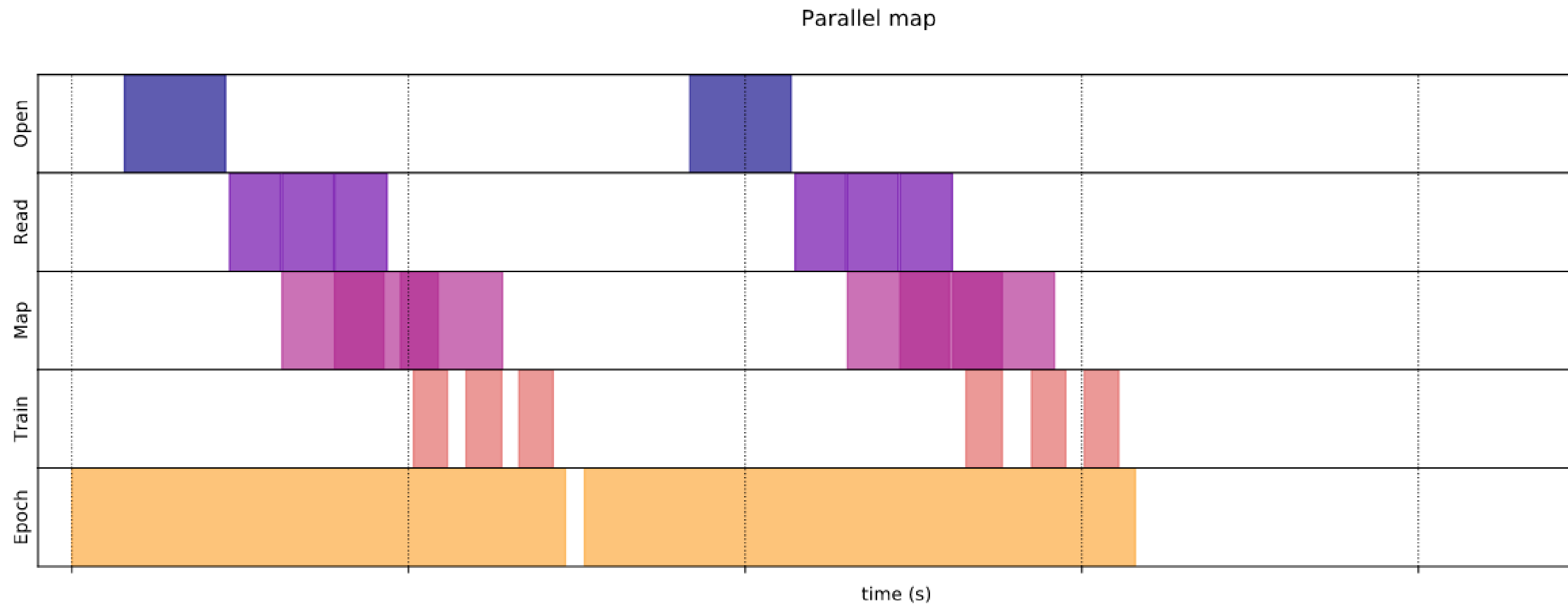
https://www.tensorflow.org/guide/data_performance

Sequential vs parallel mapping

```
benchmark(
    ArtificialDataset()
    .map(mapped_function)
)
```

```
benchmark(
    ArtificialDataset()
    .map(
        mapped_function,
        num_parallel_calls=tf.data.AUTOTUNE
    )
)
```



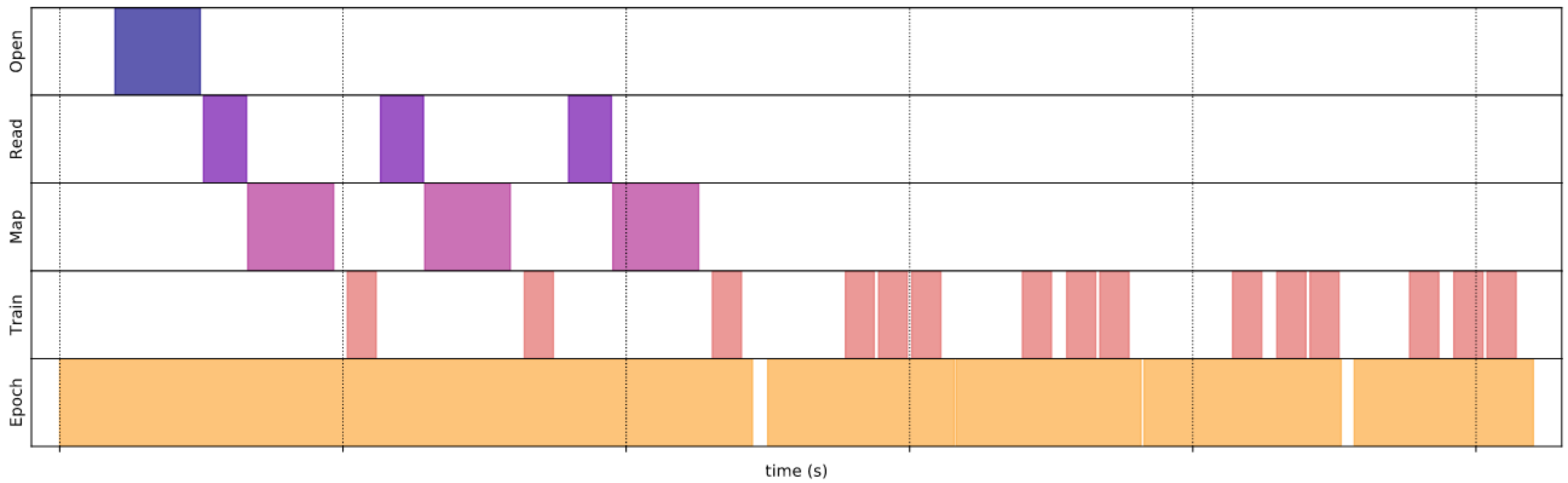


https://www.tensorflow.org/guide/data_performance

Caching

```
benchmark(
    ArtificialDataset()
    .map(  # Apply time consuming operations before cache
        mapped_function
    ).cache(
    ),
    5
)
```

Cached dataset

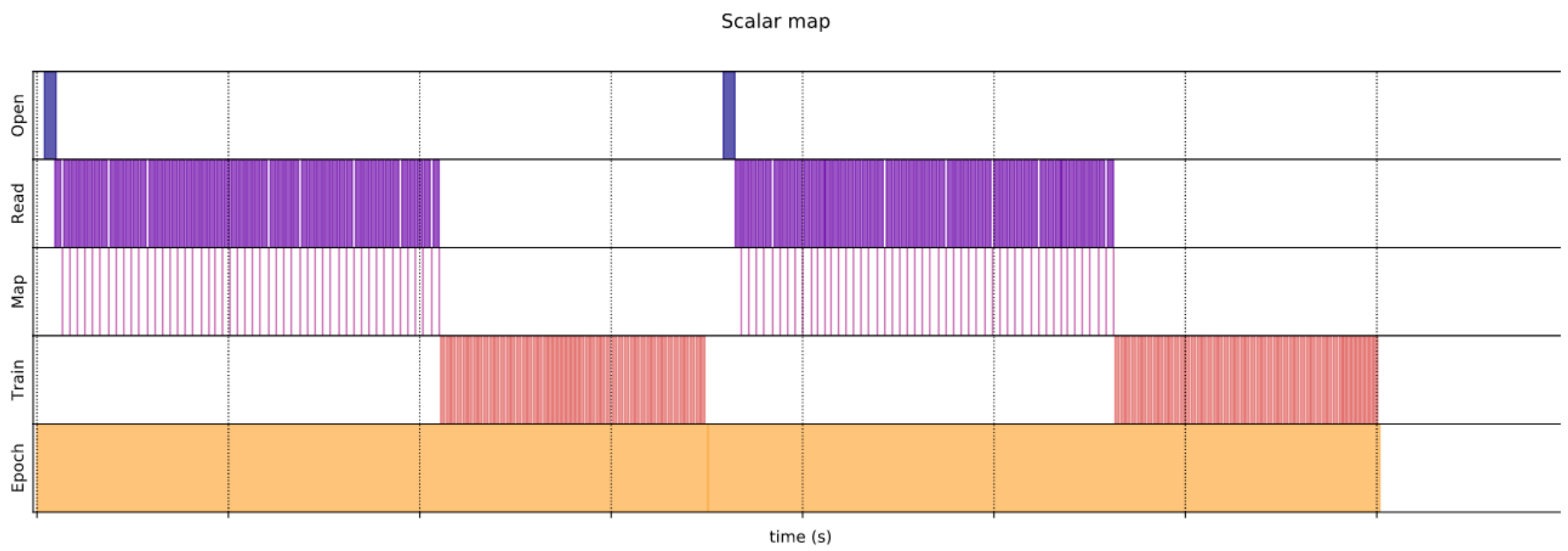


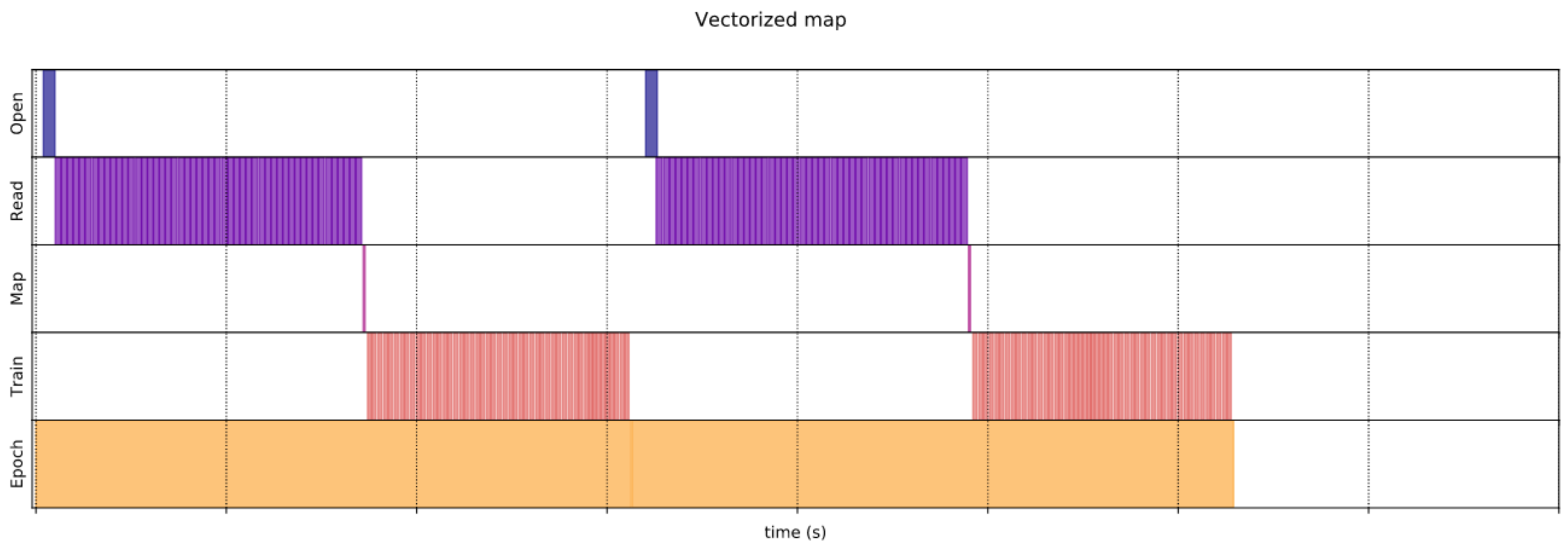
https://www.tensorflow.org/guide/data_performance

Scalar vs Vectorized mapping

```
fast_benchmark(
    fast_dataset
    # Apply function one item at a time
    .map(increment)
    # Batch
    .batch(256)
)
```

```
fast_benchmark(
    fast_dataset
    .batch(256)
    # Apply function on a batch of items
    # The tf.Tensor.__add__ method already handle batches
    .map(increment)
)
```





https://www.tensorflow.org/guide/data_performance

TfDS

```
import tensorflow_datasets as tfds

# Download the dataset and create a tf.data.Dataset
ds, info = tfds.load("mnist", split="train", with_info=True)

# Access relevant metadata with DatasetInfo
print(info.splits["train"].num_examples)
print(info.features["label"].num_classes)

# Build your input pipeline
ds = ds.batch(128).repeat(10)

# And get NumPy arrays if you'd like
for ex in tfds.as_numpy(ds):
  np_image, np_label = ex["image"], ex["label"]
```

https://colab.research.google.com/github/tensorflow/datasets/blob/master/docs/overview.ipynb

Why Distributed Training

- An easy way to distribute TensorFlow training.
- Goals:
  - Easy to use - minimal code changes
  - Great out-of-the-box performance
  - Versatile - works with different architectures, hardware and APIs

Credit: Jiri Simsa

Use cases of tf.distribute.strategy

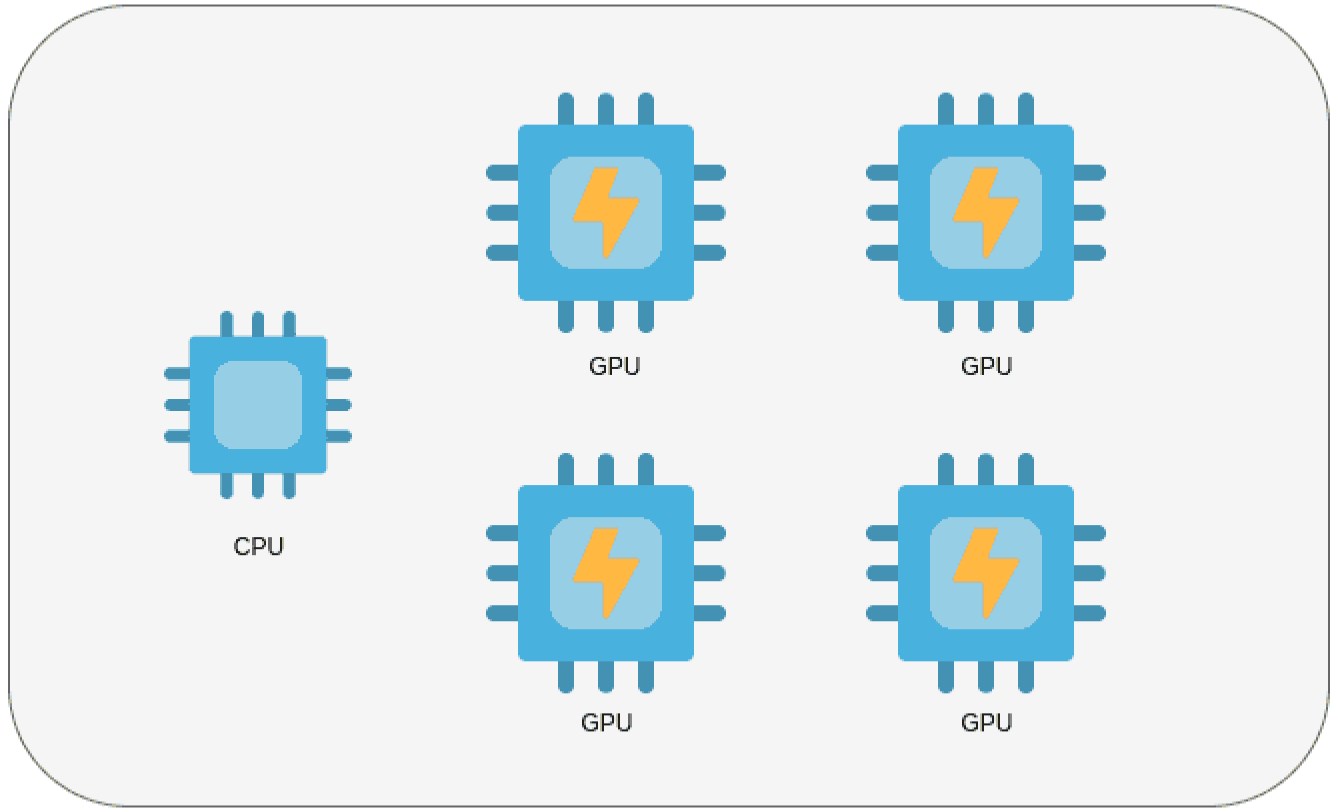
- Distribute my model using Keras / Estimator API
- Distribute my model using a custom training loop
- Make my layer / library / infrastructure distribute-aware
- Make a new strategy

Different API surfaces for each

https://www.tensorflow.org/guide/distributed_training

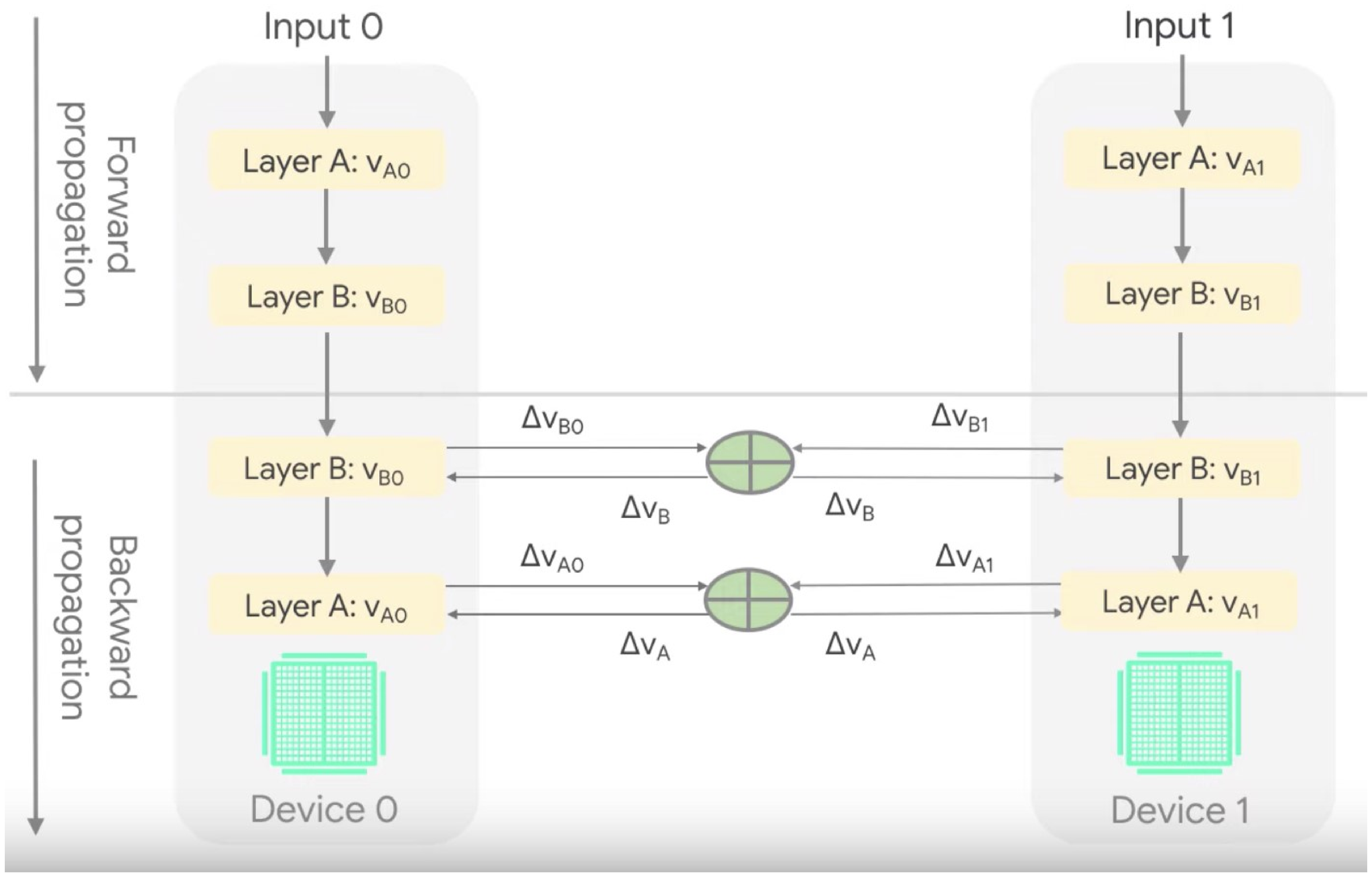
Multi-GPU All-reduce sync training

- All-reduce synchronous training for multi-GPU
  - replicas are run in lock-step, synchronizing gradients at each step
  - Variables are mirrored on each GPU
  - All-reduce: network efficient way to aggregate gradients



https://theaisummer.com/distributed-training/

Synchronous training



Credit: Jiri Simsa

MirroredStrategy

```
import tensorflow as tf


strategy = tf.distribute.MirroredStrategy()
strategy = tf.distribute.MirroredStrategy(devices=["gpu:0", "gpu:1"])
strategy = tf.distribute.MirroredStrategy(
    cross_device_op=tf.distribute.NcclAllReduce(num_packs=2))
```

Credit: Jiri Simsa

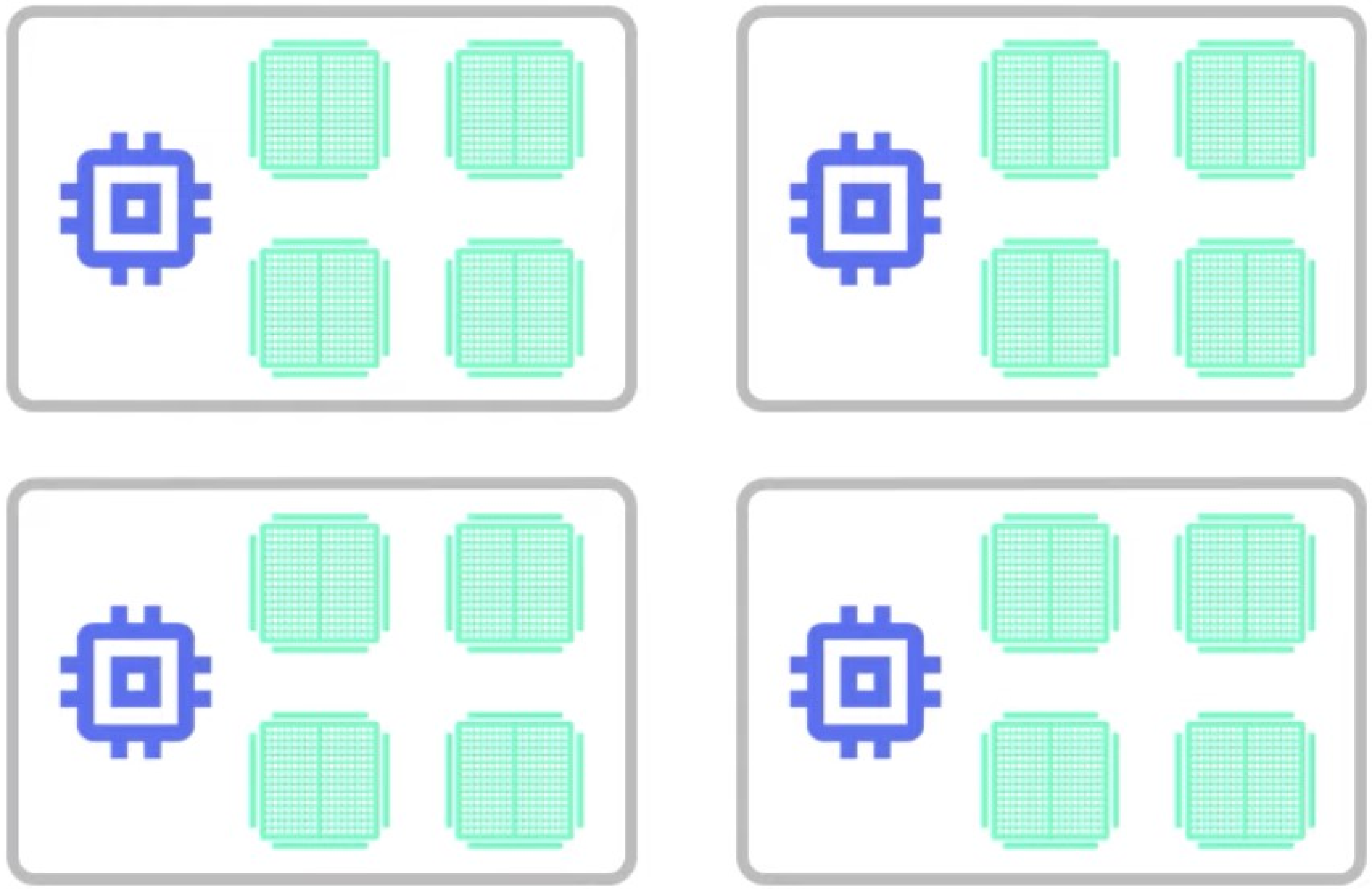
Use the strategy with Keras API

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Multi worker all-reduce sync training

- Uses new collective ops
- Workers are run in lock-step,
- synchronizing gradients at each step



Credit: Jiri Simsa

Multi worker all-reduce sync training

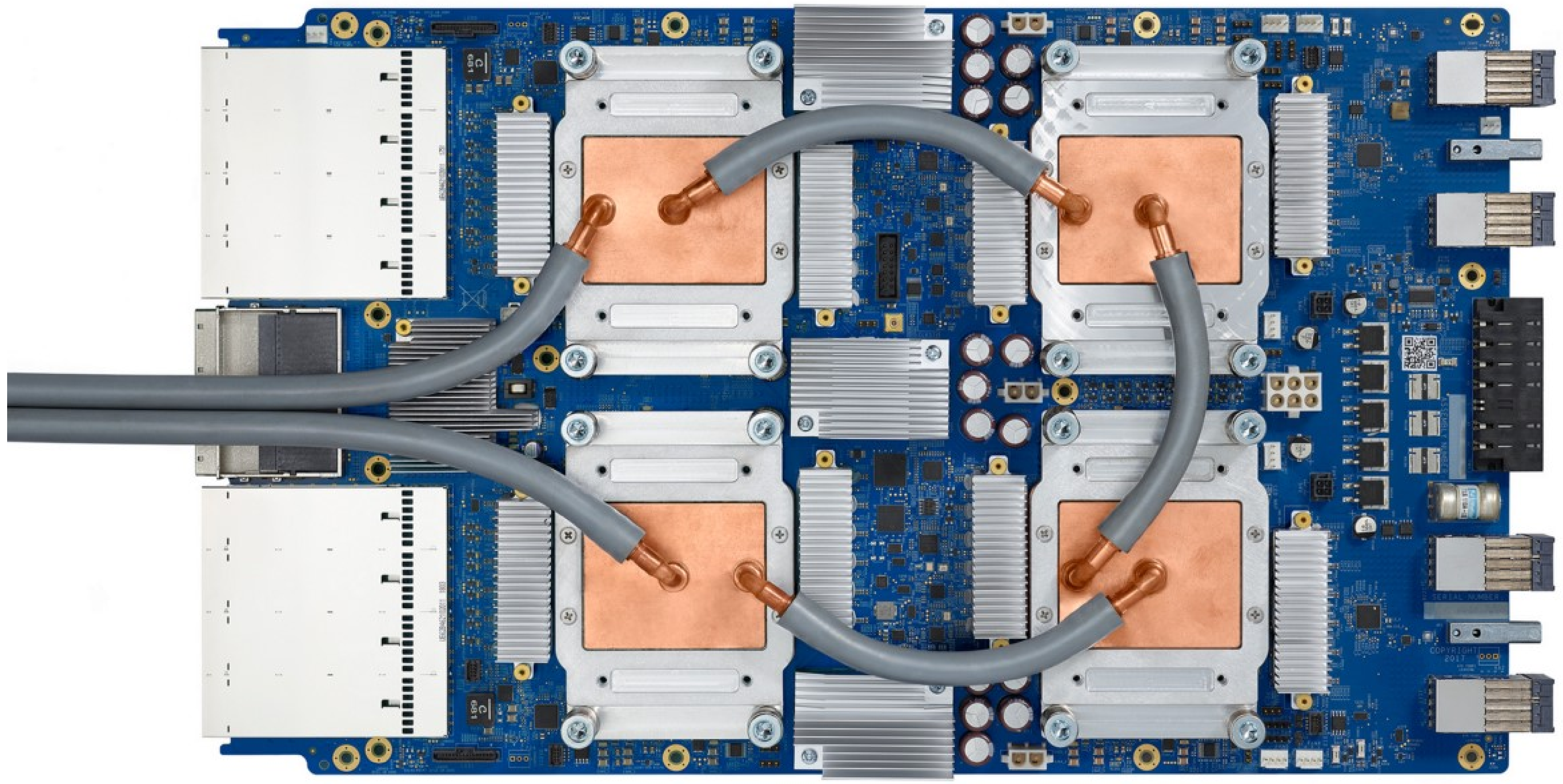
```
import tensorflow as tf


strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy(
    tf.distribute.experimental.CollectiveCommunication.NCCL)
```

Credit: Jiri Simsa

All-reduce sync training for TPUs

- Similar to MirroredStrategy
- Uses cross-replica-sum on TPUs to do all reduce

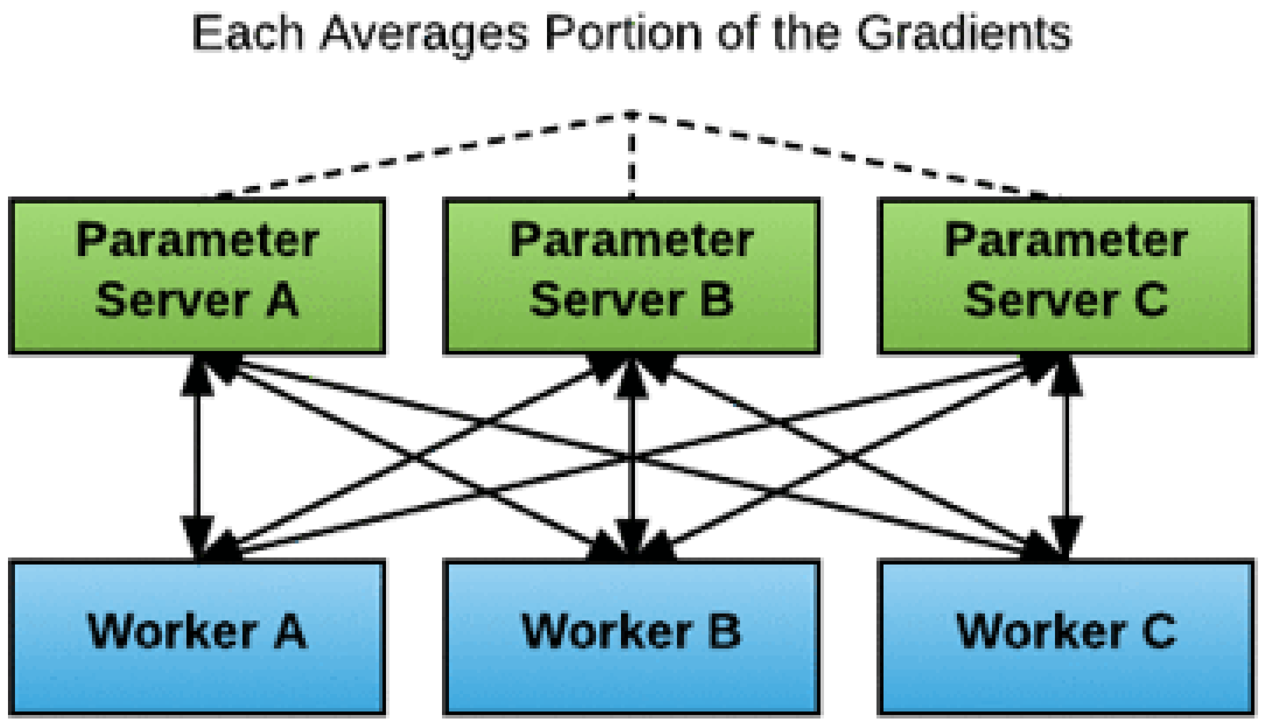


https://medium.com/@decoded_cipher/tensor-processing-units-both-history-and-applications-b3479d92a61d

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='')
tf.config.experimental_connect_to_cluster(resolver)
tf.tpu.experimental.initialize_tpu_system(resolver)
strategy = tf.distribute.experimental.TPUStrategy(resolver)
```

Credit: Jiri Simsa

Parameter Servers and Workers

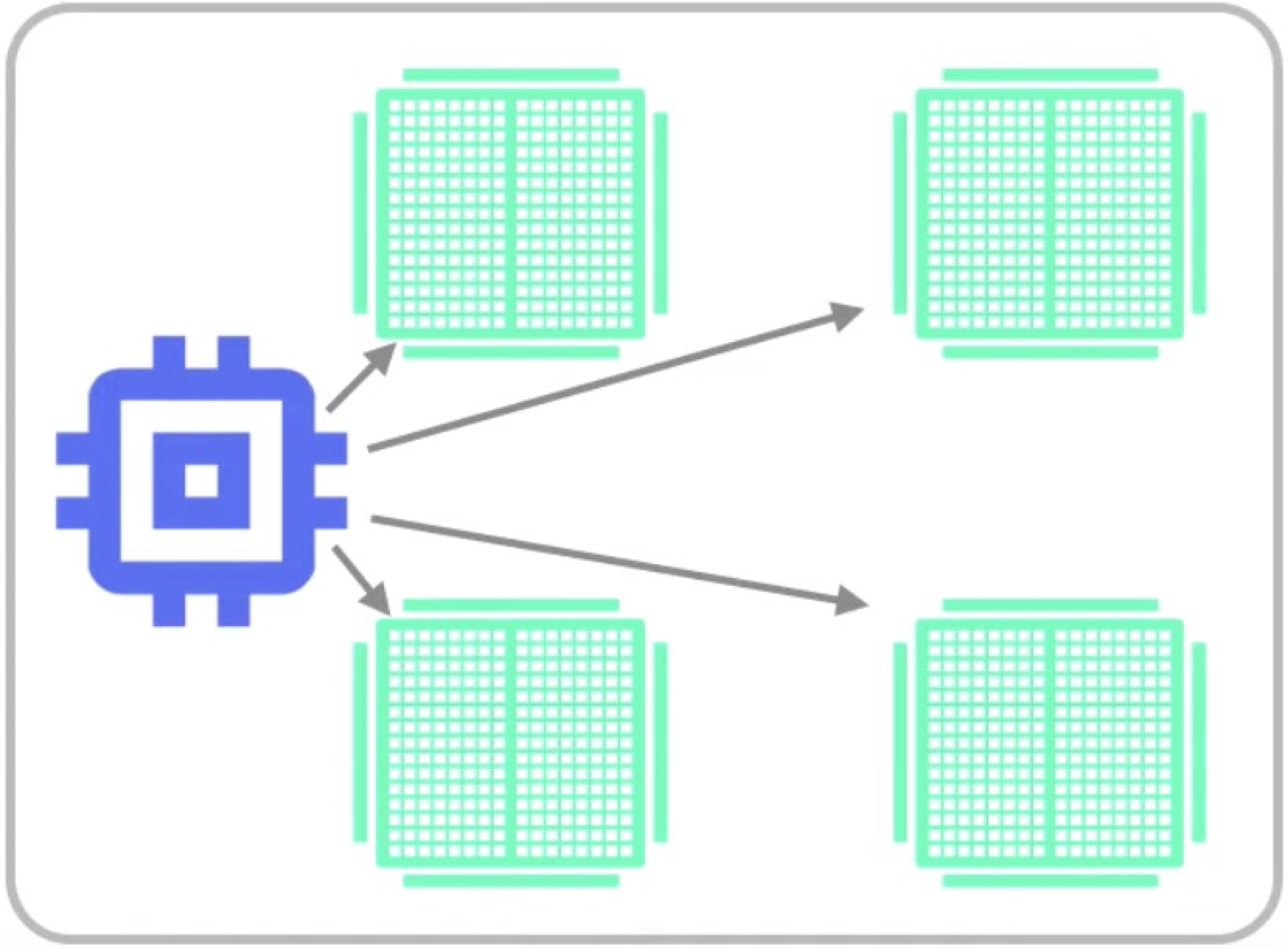



```
ps_strategy = tf.distribute.experimental.ParameterServerStrategy()
parameter_server_strategy = tf.distribute.experimental.ParameterServerStrategy()
```

Credit: AWS

Central storage

- 1-machine multi-GPU
  - one copy of each variable on CPU
  - one replica of the model per GPU
  - Can handle embeddings that wont fit on a GPU



```
import tensorflow as tf


strategy = tf.distribute.experimental.CentralStorageStrategy()
```

Credit: Jiri Simsa

Reference code

- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/distribute/custom_training.ipynb
- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/distribute/multi_worker_with_keras.ipynb
- https://github.com/python-engineer/tensorflow-course/blob/master/07_Functional_API_Project.ipynb

Thank You