TensorFlow 2.0 Tutorial: Part #5

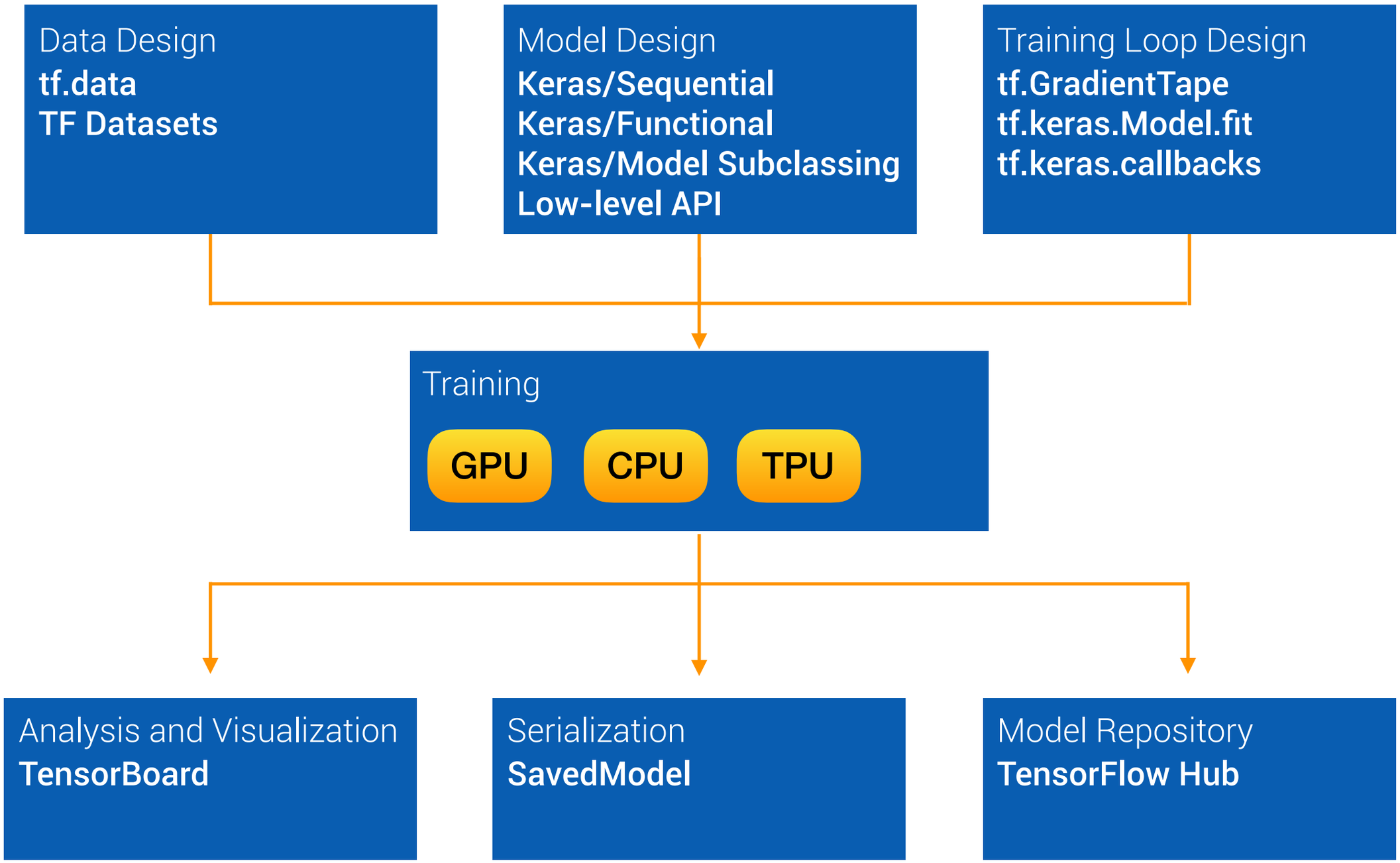
Data Pipeline APIs (tf.data)

Iran University of Science and Technology (IUST)
Department of Computer Engineering

URL:

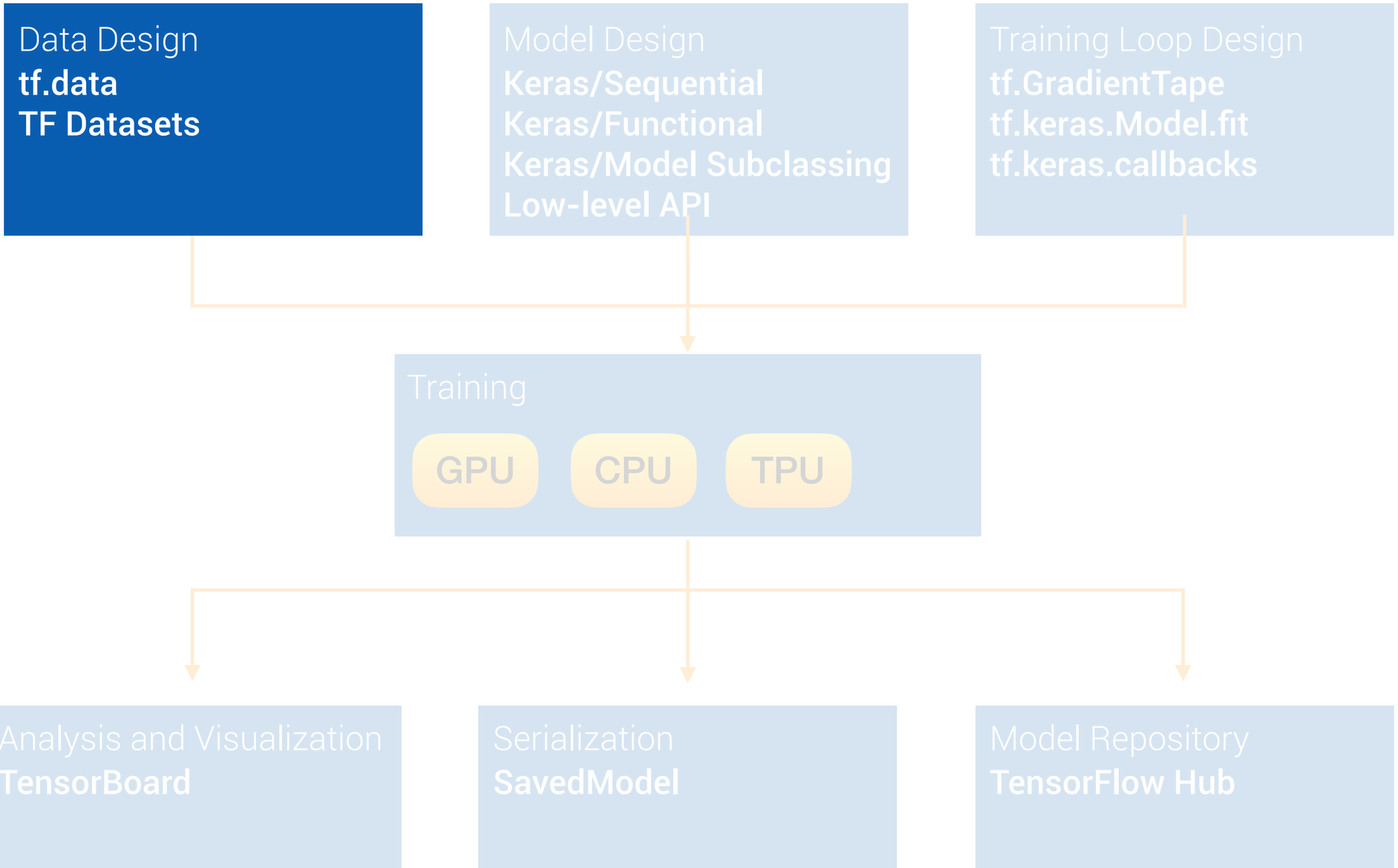
github.com/iust-deep-learning/tensorflow-2-tutorial/tree/master/part_05_tf_data

TensorFlow Overview



[source]

TensorFlow Overview



Outline

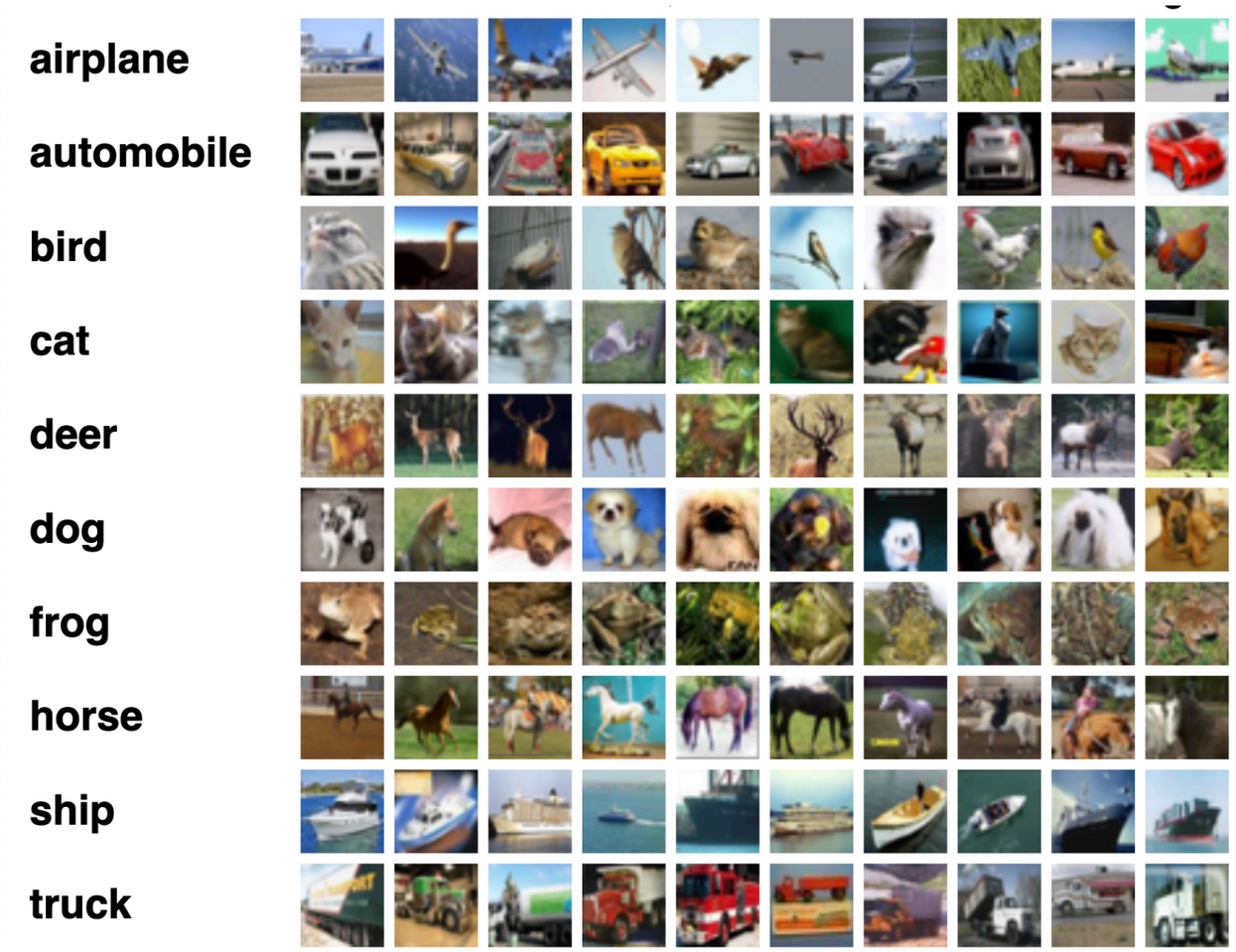
- **Old Style vs. tf.data**
  - **tf.data motivation**
- **tf.data.Dataset Mechanism**
  - **Structured Elements**
- **Using tf.data.Dataset**
  - **Data Sources**
- **Common Transformations**

Outline

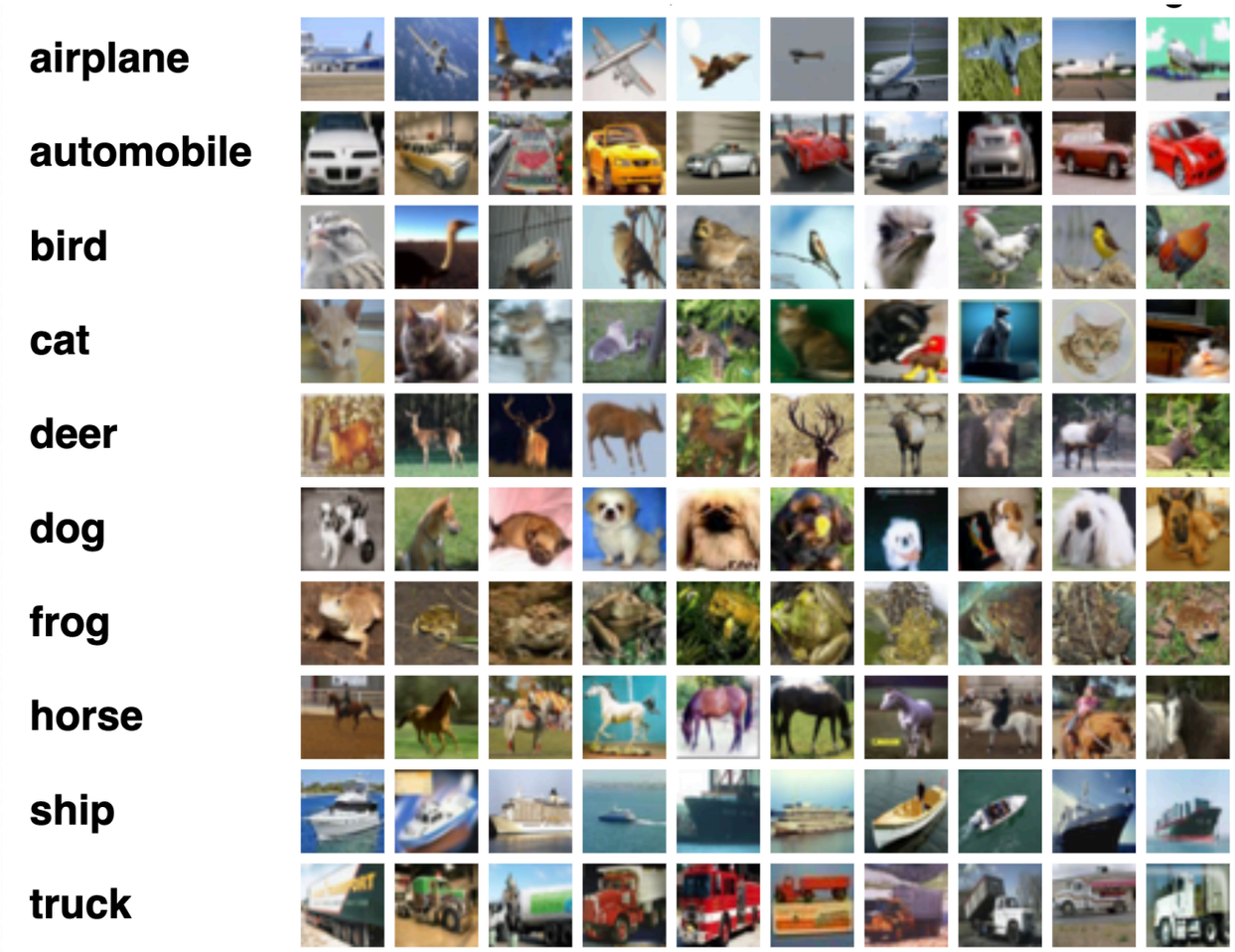
- **Old Style vs. tf.data**
- tf.data.Dataset Mechanism
- Using tf.data.Dataset
- Common Transformations

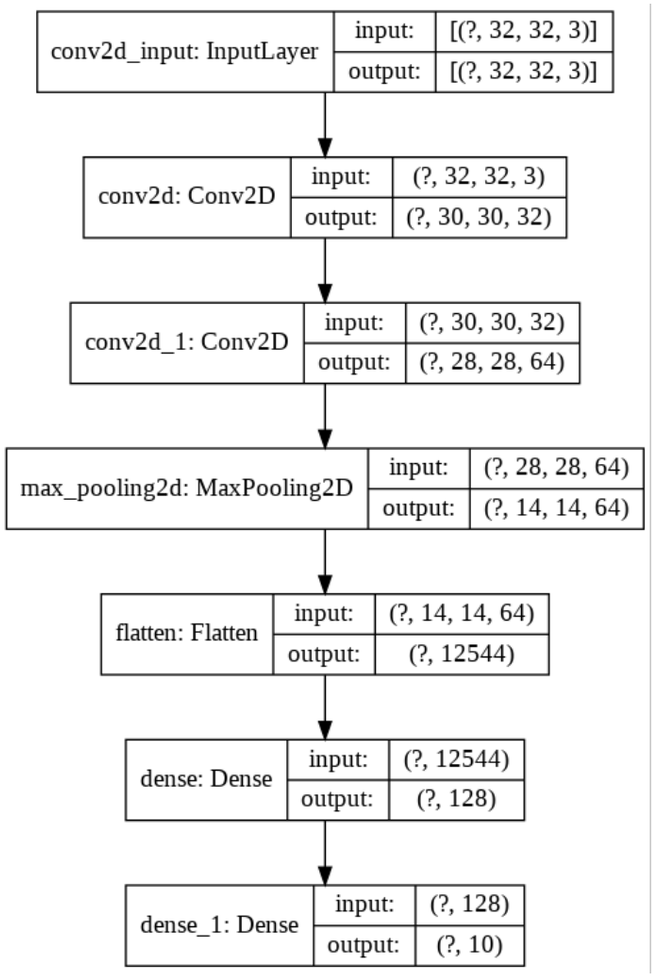
- tf.data motivation
- Structured Elements
- Data Sources

Case Study: CIFAR-10 Classifier



Case Study: CIFAR-10 Classifier





Case Study: CIFAR-10 Classifier

```
model = tf.keras.Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Case Study: CIFAR-10 Classifier

```
model = tf.keras.Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Case Study: CIFAR-10 Classifier

```
model = tf.keras.Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(data_generator, epochs=10)
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)

        # Resize the image
        image = resize(image, (32, 32))
        # Normalize the image and add noise to it
        image = normalize_and_add_noise(image)
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)

# Resize the image
image = resize(image, (32, 32))
# Normalize the image and add noise to it
image = normalize_and_add_noise(image)

# Handle batching
batch.append((image, label))
if len(batch_img) == batch_size:
    yield concat(batch)
    batch.reset()
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)
        # Resize the image
        image = resize(image, (32, 32))
        # Normalize the image and add noise to it
        image = normalize_and_add_noise(image)
        # Handle batching
        batch.append((image, label))
        if len(batch_img) == batch_size:
            yield concat(batch)
            batch.reset()
```

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)
        # Resize the image
        image = resize(image, (32, 32))
        # Normalize the image and add noise to it
        image = normalize_and_add_noise(image)
        # Handle batching
        batch.append((image, label))
        if len(batch_img) == batch_size:
            yield concat(batch)
            batch.reset()
```

Extract

Transform

Load

Case Study: CIFAR-10 Classifier

```
def data_generator():
    shuffle(data) # contains file names and labels
    batch = []
    for image_path, label in data:
        # Read the image from disk
        image = imread(image_path)
        # Resize the image
        image = resize(image, (32, 32))
        # Normalize the image and add noise to it
        image = normalize_and_add_noise(image)
        # Handle batching
        batch.append((image, label))
        if len(batch_img) == batch_size:
            yield concat(batch)
            batch.reset()
```

Extract (IO)

Transform (CPU)

Load (Memory)

What is the problem with this naive data-generator?

What is the problem with this naive data-generator?

- It blocks the training loop (and GPU) while reading images from the Disk.
- It does not support multi threading.

Enter the `tf.data`

Outline

- **Old Style vs. tf.data**
  - tf.data motivation
- tf.data.Dataset Mechanism
  - Structured Elements
- Using tf.data.Dataset
  - Data Sources
- Common Transformations

Outline

- Old Style vs. tf.data
- tf.data.Dataset Mechanism
- Using tf.data.Dataset
- Common Transformations
- **tf.data motivation**
- Structured Elements
- Data Sources

Naive

CPU

GPU

Naive

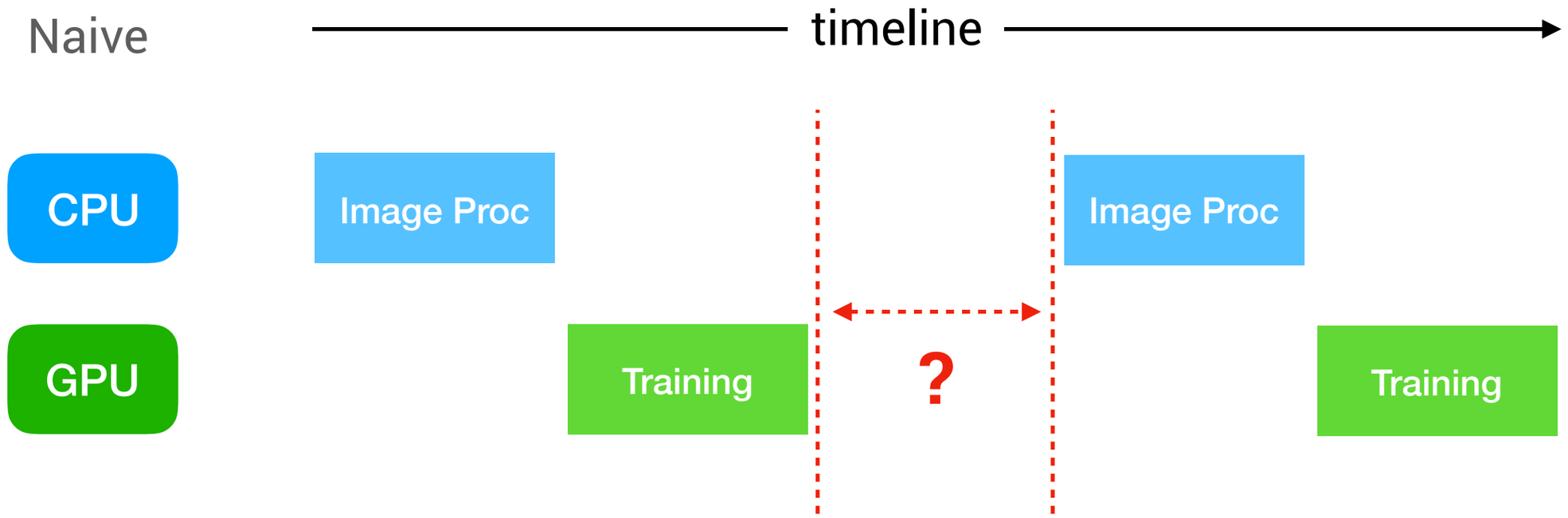
CPU

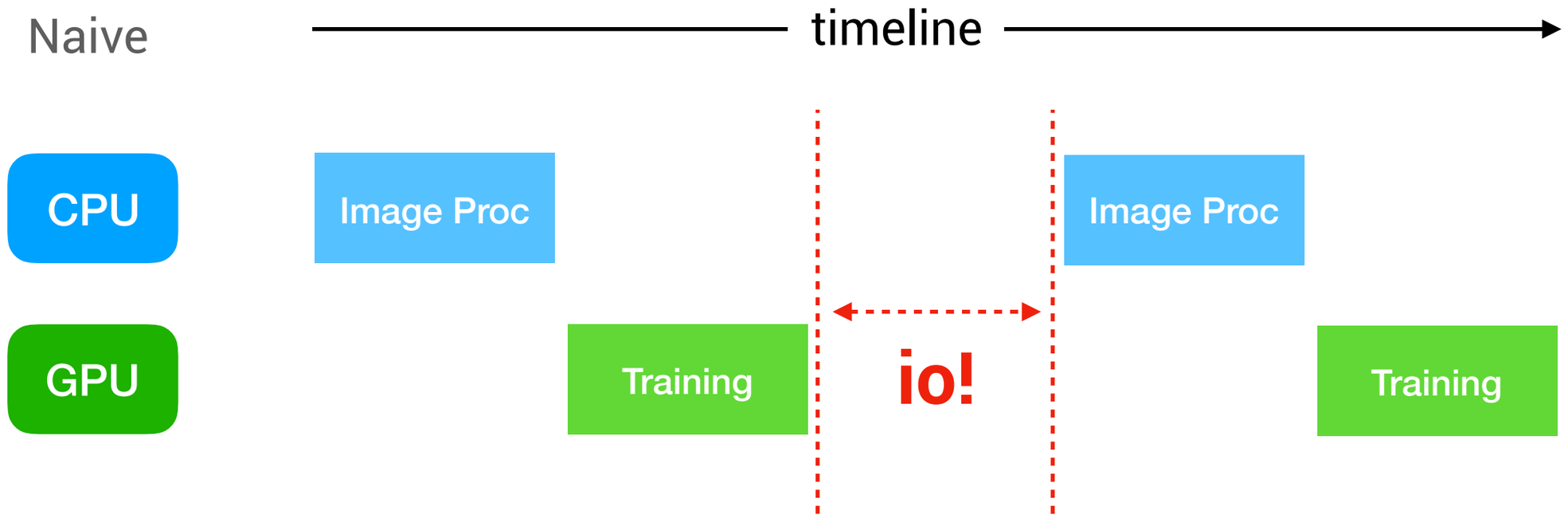
Image Proc

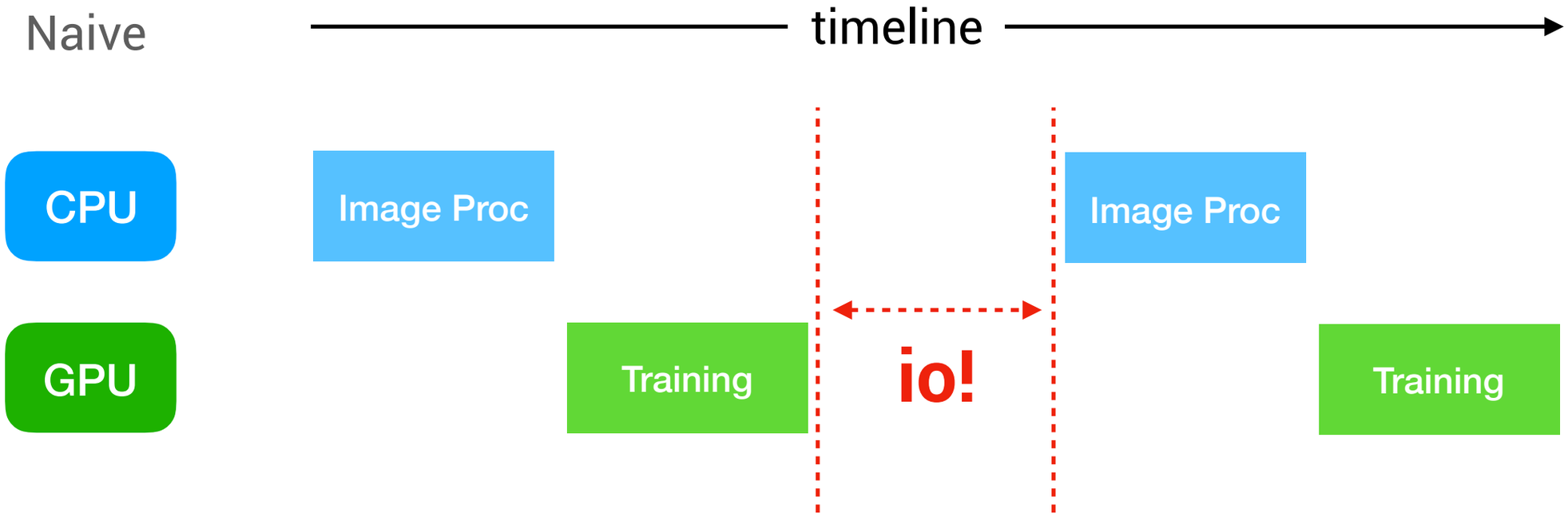
GPU

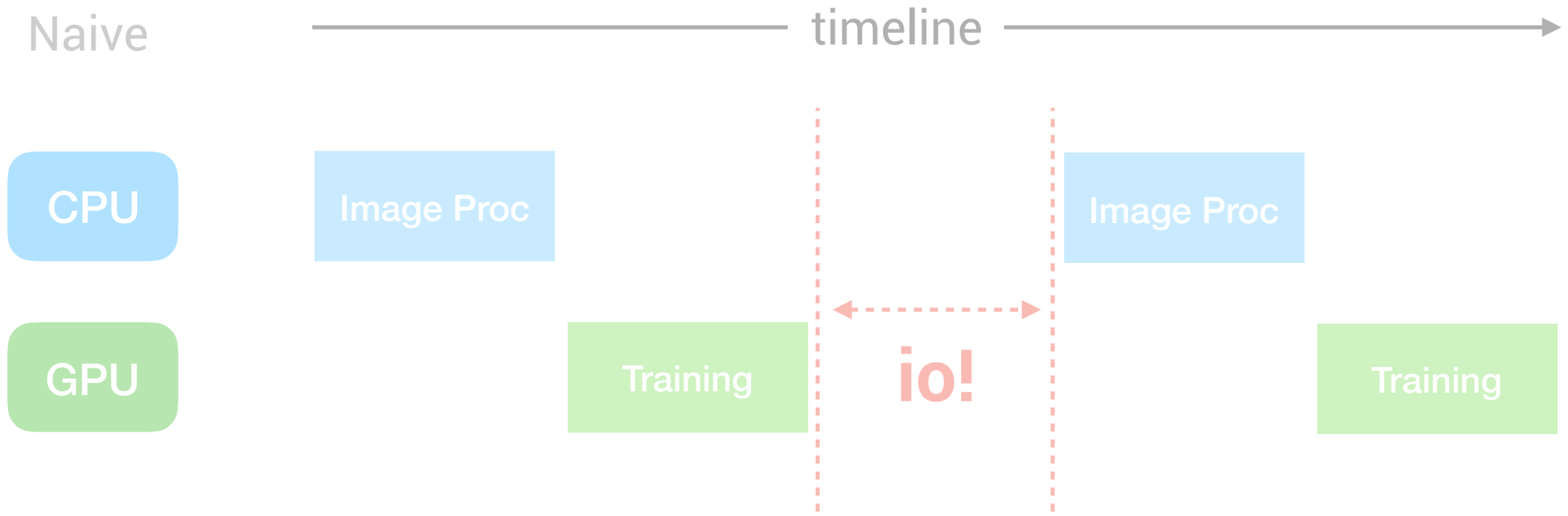

Naive
timeline
CPU
Image Proc
GPU
Training


Naive
timeline
CPU
Image Proc
Image Proc
GPU
Training
Training


Naive
timeline
CPU
Image Proc
Image Proc
GPU
Training
?
Training


Naive
timeline
CPU
Image Proc
Image Proc
GPU
Training
io!
Training


Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training


Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training

Naive

timeline

CPU

GPU

Image Proc

Training

io!

Image Proc

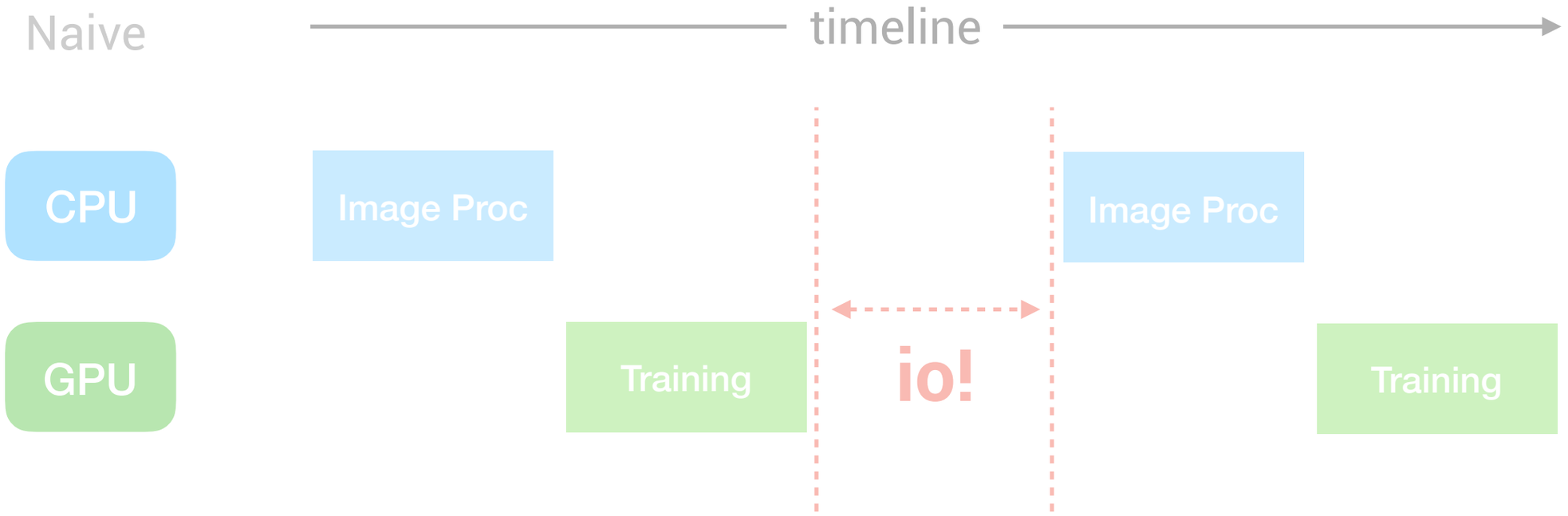
Training

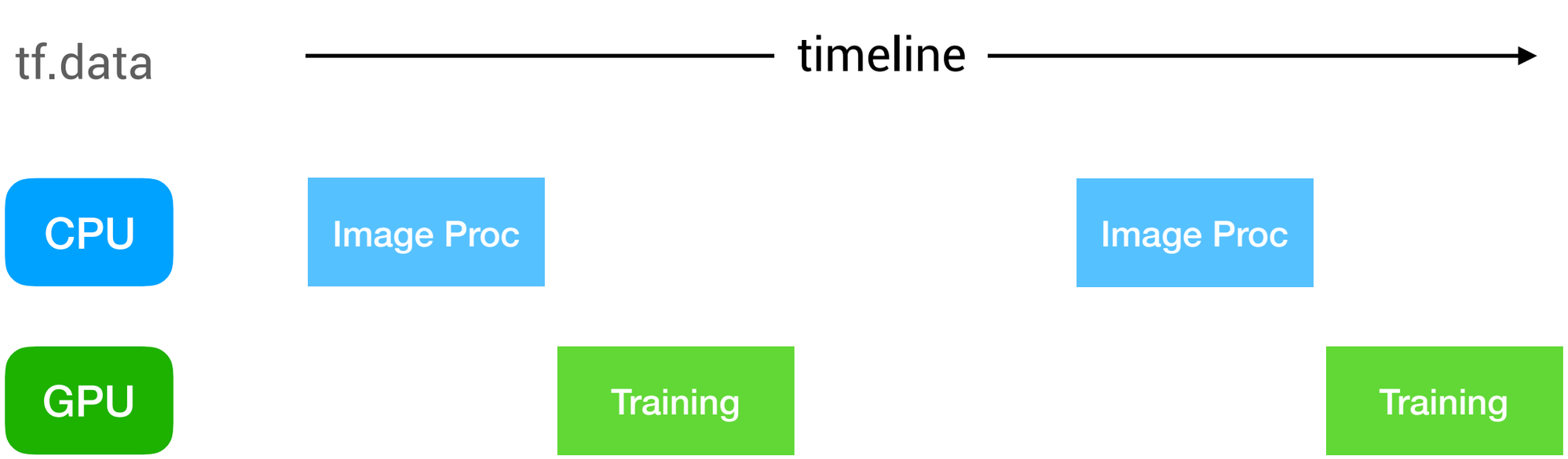
tf.data

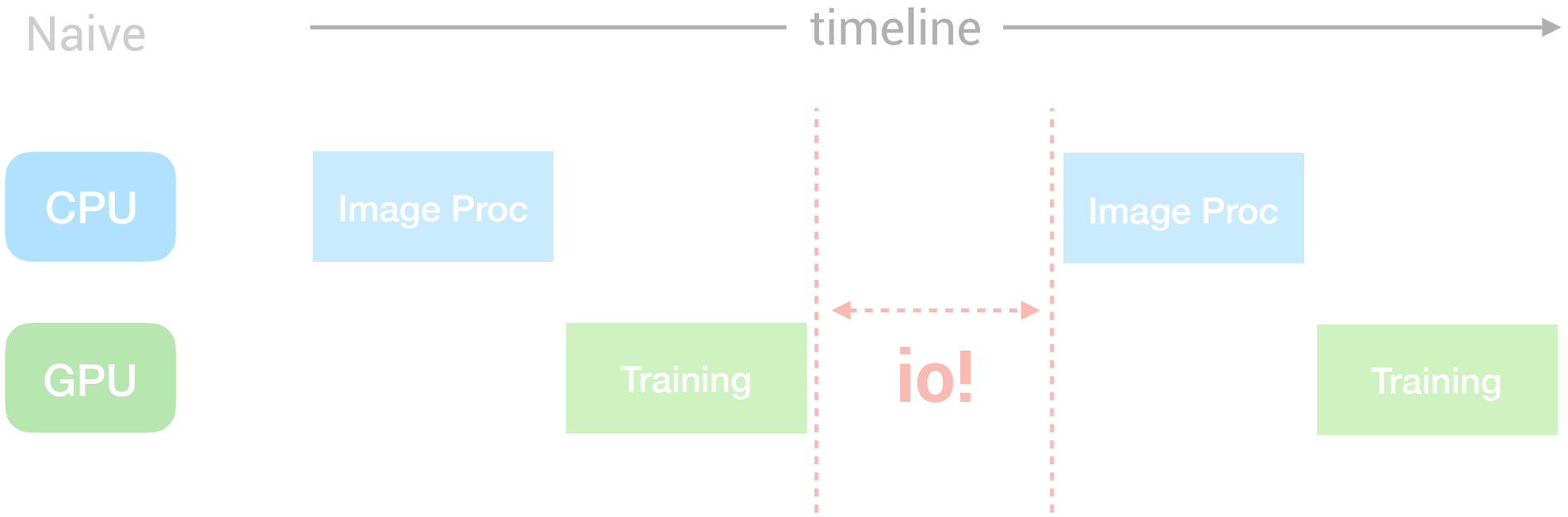
timeline

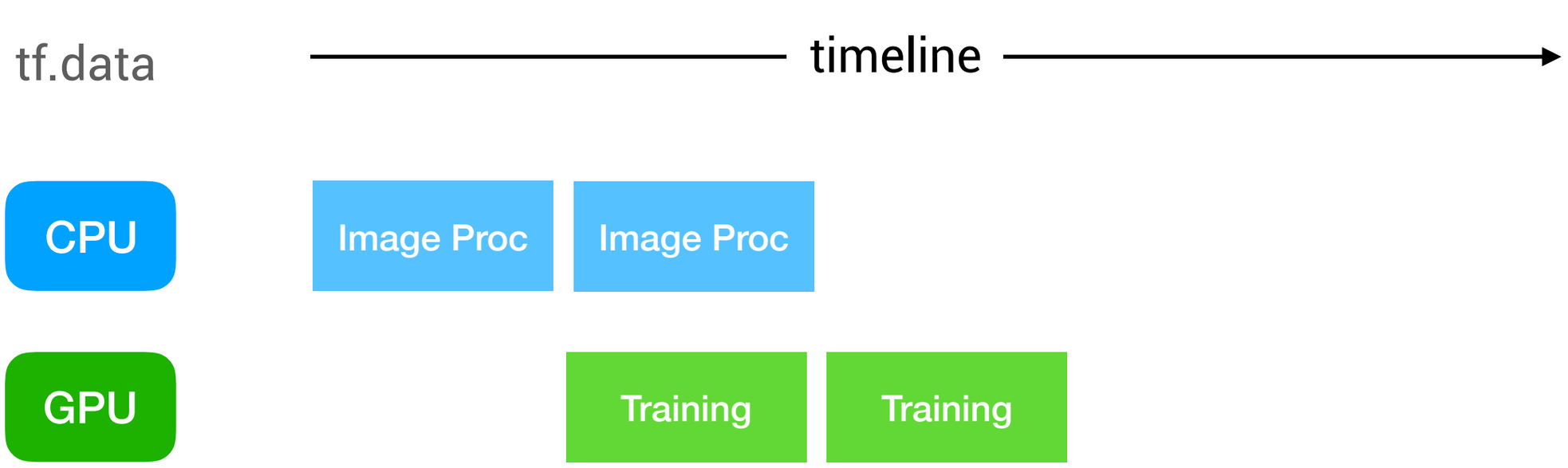
CPU

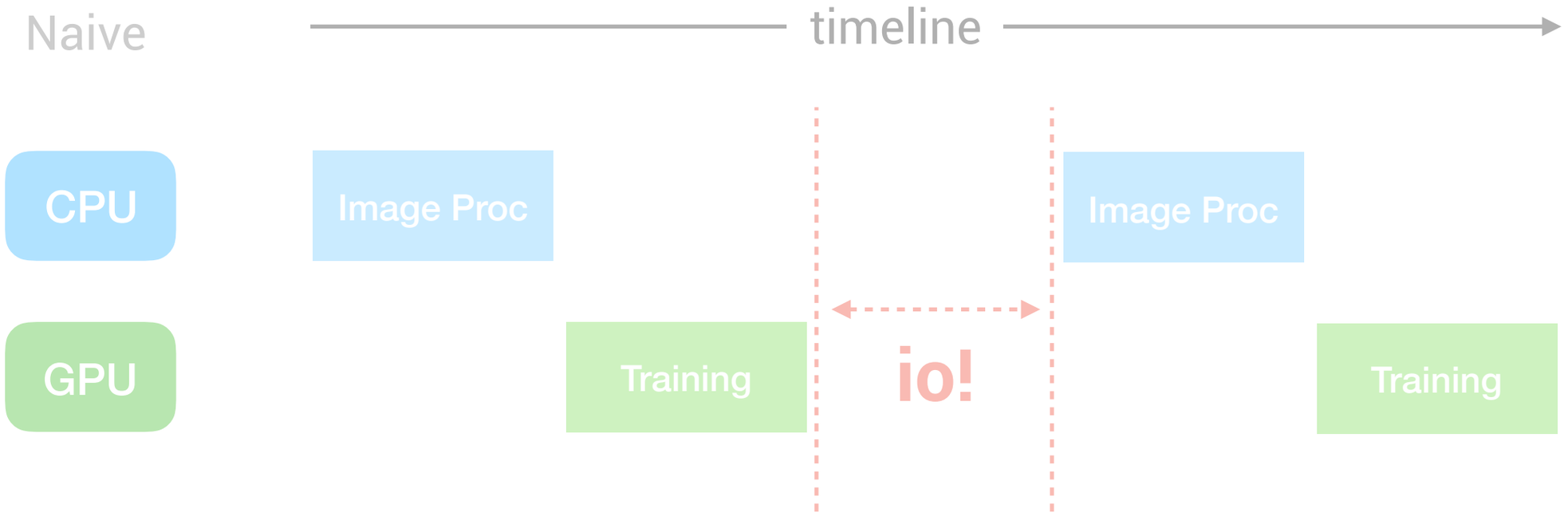
GPU

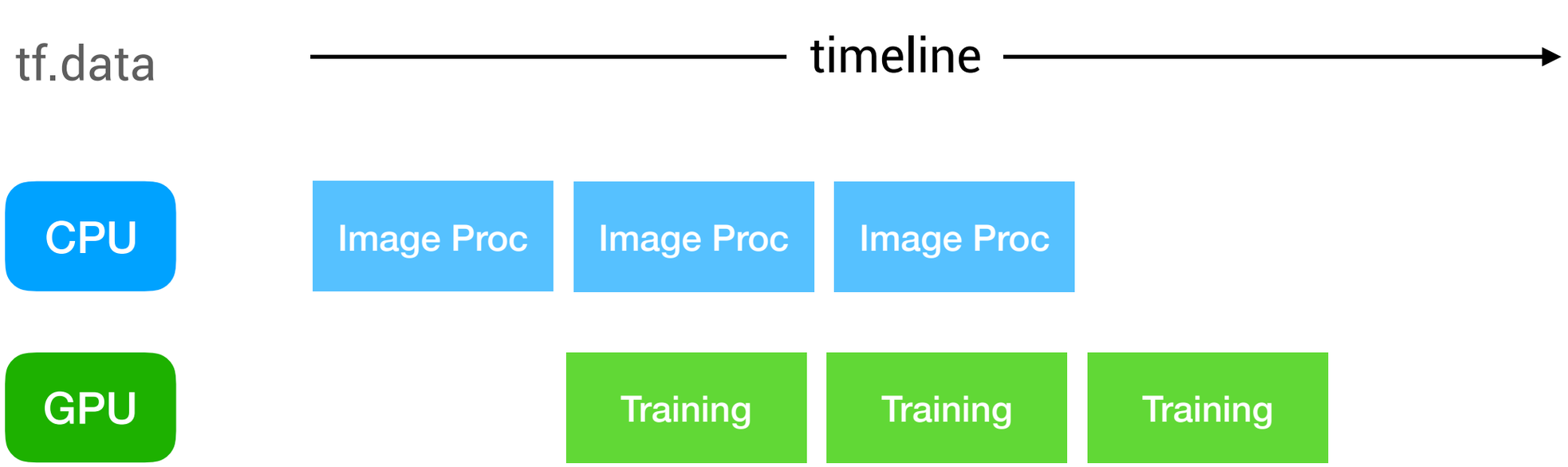

Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training

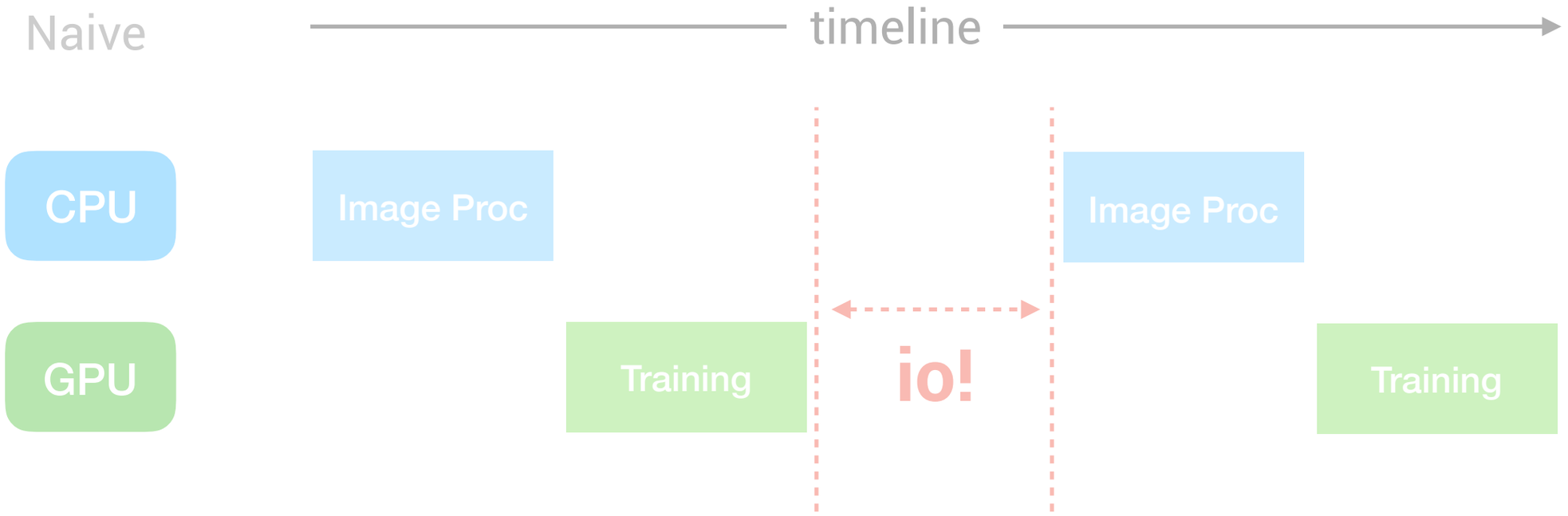

tf.data
timeline
CPU
GPU
Image Proc
Training
Image Proc
Training

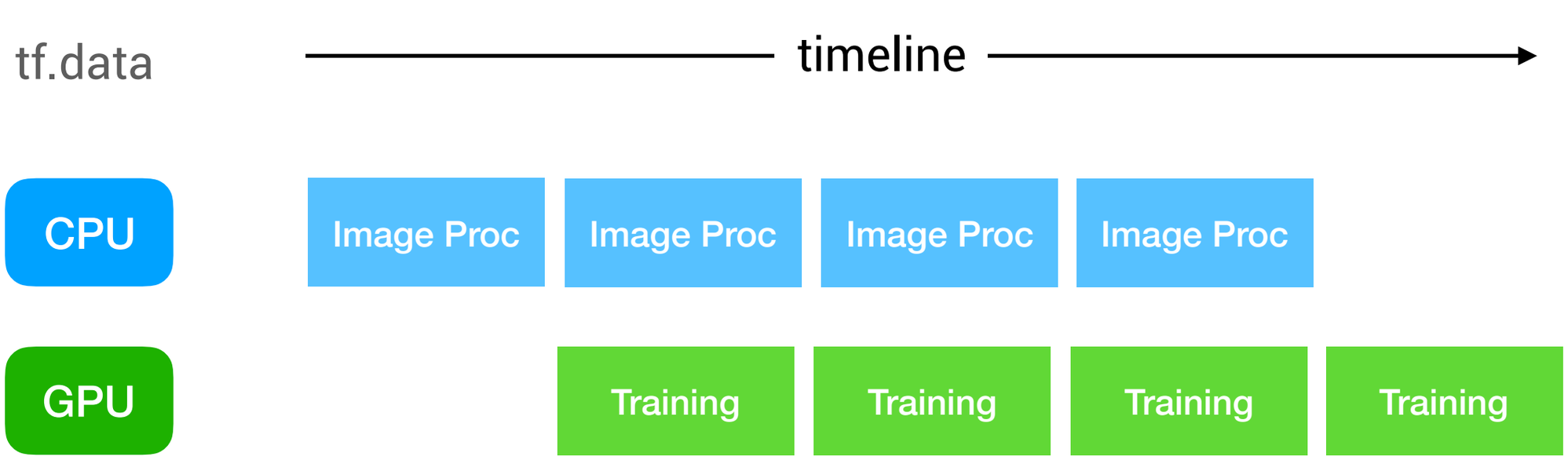

Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training


tf.data
timeline
CPU
GPU
Image Proc
Image Proc
Training
Training


Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training


tf.data
timeline
CPU
GPU
Image Proc
Image Proc
Image Proc
Training
Training
Training


Naive
timeline
CPU
GPU
Image Proc
Training
io!
Image Proc
Training


tf.data
timeline
CPU
GPU
Image Proc
Image Proc
Image Proc
Image Proc
Training
Training
Training
Training

tf.data: TensorFlow Input Pipeline

Extract

✓ read data from memory / storage
✓ parse file format

Transform

✓ text vectorization
✓ image transformation
✓ video temporal sampling

Load

✓ transfer data to the accelerator

tf.data: TensorFlow Input Pipeline

Extract

✓ read data from memory / storage
✓ parse file format

Transform

✓ text vectorization
✓ image transformation
✓ video temporal sampling

Load

✓ transfer data to the accelerator

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
```

Data Source

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
```

Other
Transformation

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)

def read_image_and_label(filepath):
  # label, filepath: 0_28382.jpg
  parts = tf.strings.split(filepath, "_")
  label = tf.strings.to_number(parts[0], out_type=tf.int32)
  # image
  raw_bytes = tf.io.read_file(filepath)
  return raw_bytes, label
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)

def process_image(raw_bytes, label):
  img = tf.image.decode_jpeg(raw_bytes, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)
  img = tf.image.resize(img, [32, 32])

  img = img / 255.0
  img += tf.random.normal(img.shape, stdev=0.1)

  return img, label
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

We need to find the optimal values for X, Y, and Z.

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
X = Y = Z = AUTOTUNE

dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
dataset = dataset.prefetch(buffer_size=X) # Enable pipelining
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
X = Y = Z = AUTOTUNE

dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg') \
            .shuffle(NUM_TOTAL_IMAGES) \
            .map(read_image_and_label, num_parallel_calls=y) \
            .map(process_image, num_parallel_calls=Z) \
            .batch(batch_size=64) \
            .prefetch(buffer_size=X) # Enable pipelining
```

Outline

- Old Style vs. tf.data
- tf.data.Dataset Mechanism
- Using tf.data.Dataset
- Common Transformations
- **tf.data motivation**
- Structured Elements
- Data Sources

Outline

Old Style vs. tf.data

tf.data.Dataset Mechanism

Using tf.data.Dataset

Common Transformations

tf.data motivation

Structured Elements

Data Sources

tf.data.Dataset Mechanism

`tf.data.Dataset`: abstraction that represents a sequence of elements, in which each element consists of one or more components.

tf.data.Dataset Mechanism

`tf.data.Dataset`: abstraction that represents a sequence of elements, in which each element consists of one or more components.

1. To create an instance of `Dataset`, you must start with a data source.

tf.data.Dataset Mechanism

`tf.data.Dataset`: abstraction that represents a sequence of elements, in which each element consists of one or more components.

1. To create an instance of `Dataset`, you must start with a data source.
2. You can transform it into a new `Dataset` by chaining method calls on the `tf.data.Dataset` object (such as `.map(...)`, `.batch(...)`, `.filter(...)`, and etc.

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
dataset = dataset.map(process_image, num_parallel_calls=Z)
dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)


dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)


dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)

dataset = dataset.map(process_image, num_parallel_calls=Z)


dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)

dataset = dataset.batch(batch_size=64)
```

Each element could be a structure. Here, each element is **a tuple**-a pair of tensors

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)



dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)



dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)
(TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

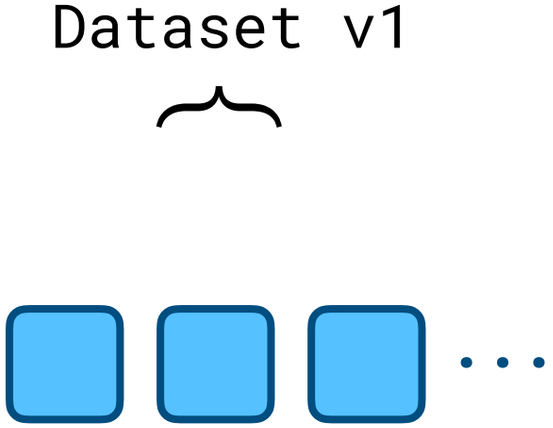
```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)
(TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.batch(batch_size=64)
```

Case Study: CIFAR-10 Classifier / Using tf.data

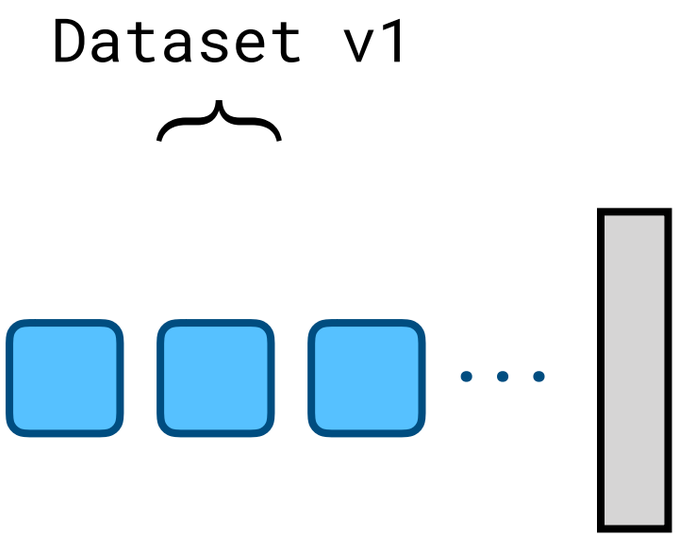
```
dataset = tf.data.Dataset.list_files('/path/to/ds/*.jpg')
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
TensorSpec(shape=(), dtype=tf.string)

dataset = dataset.map(read_image_and_label, num_parallel_calls=Y)
(TensorSpec(shape=(), dtype=tf.string),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.map(process_image, num_parallel_calls=Z)
(TensorSpec(shape=(224, 224, 3), dtype=tf.float32),
 TensorSpec(shape=(), dtype=tf.int32))

dataset = dataset.batch(batch_size=64)
(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
TensorSpec(shape=(None,), dtype=tf.int32, name=None))
```

tf.data.Dataset Mechanism

tf.data.Dataset Mechanism

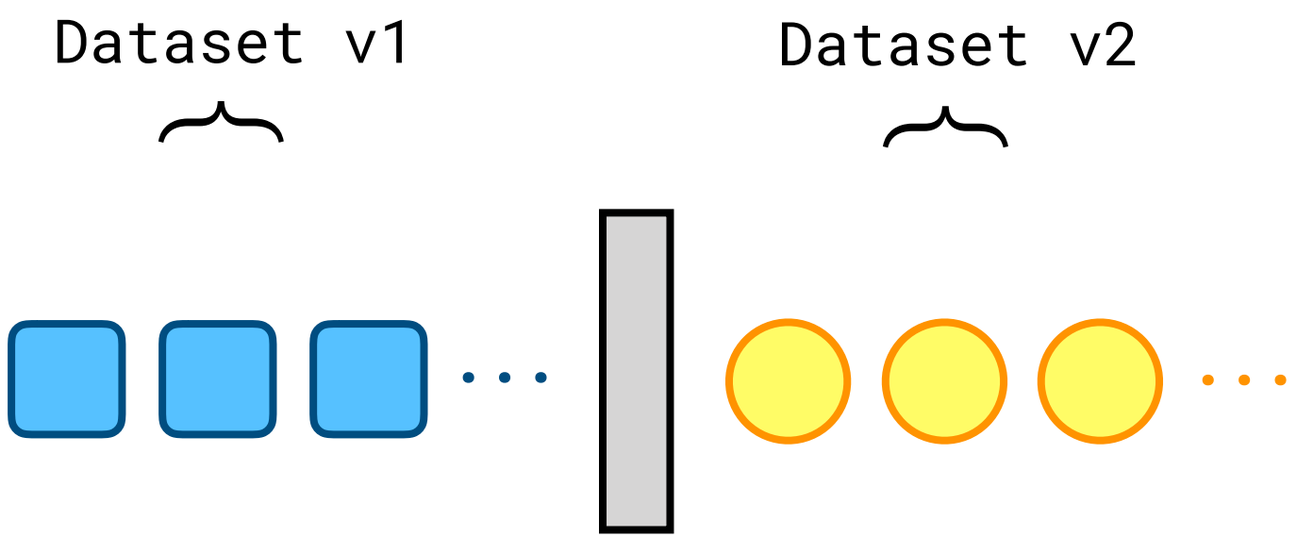


tf.data.Dataset Mechanism

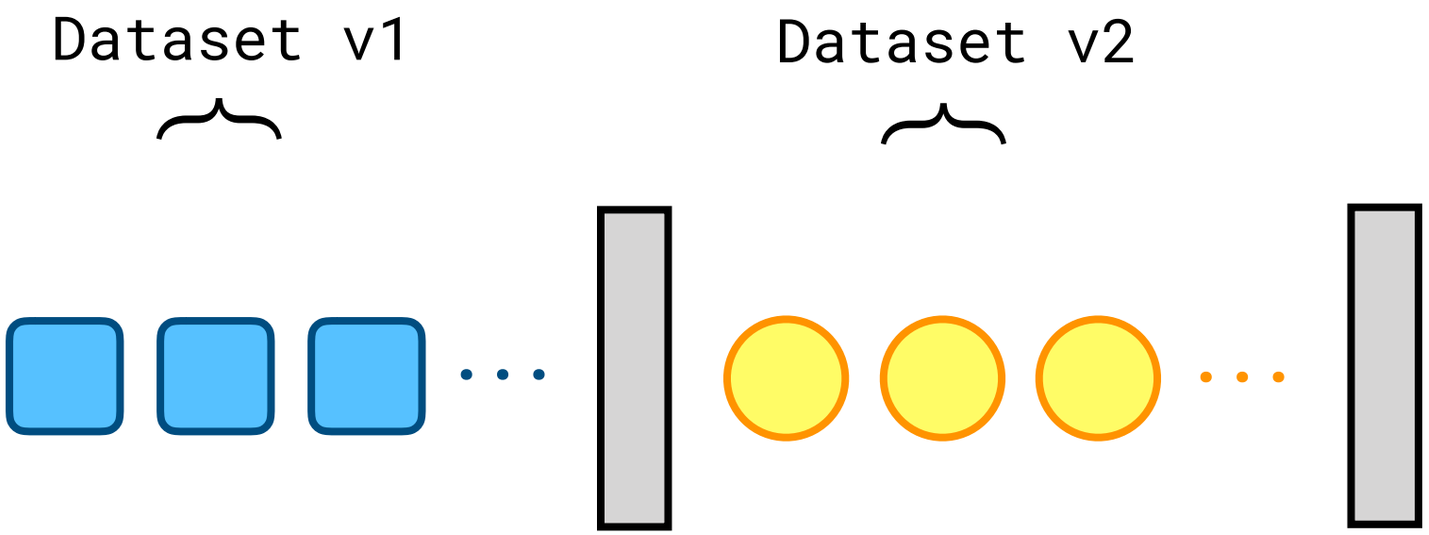


Transformation #1

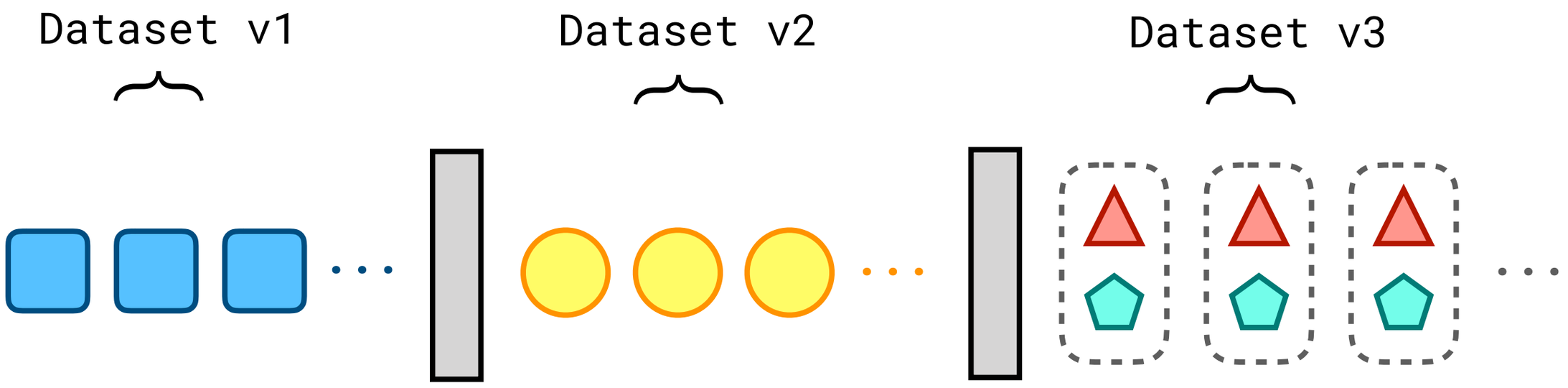
tf.data.Dataset Mechanism



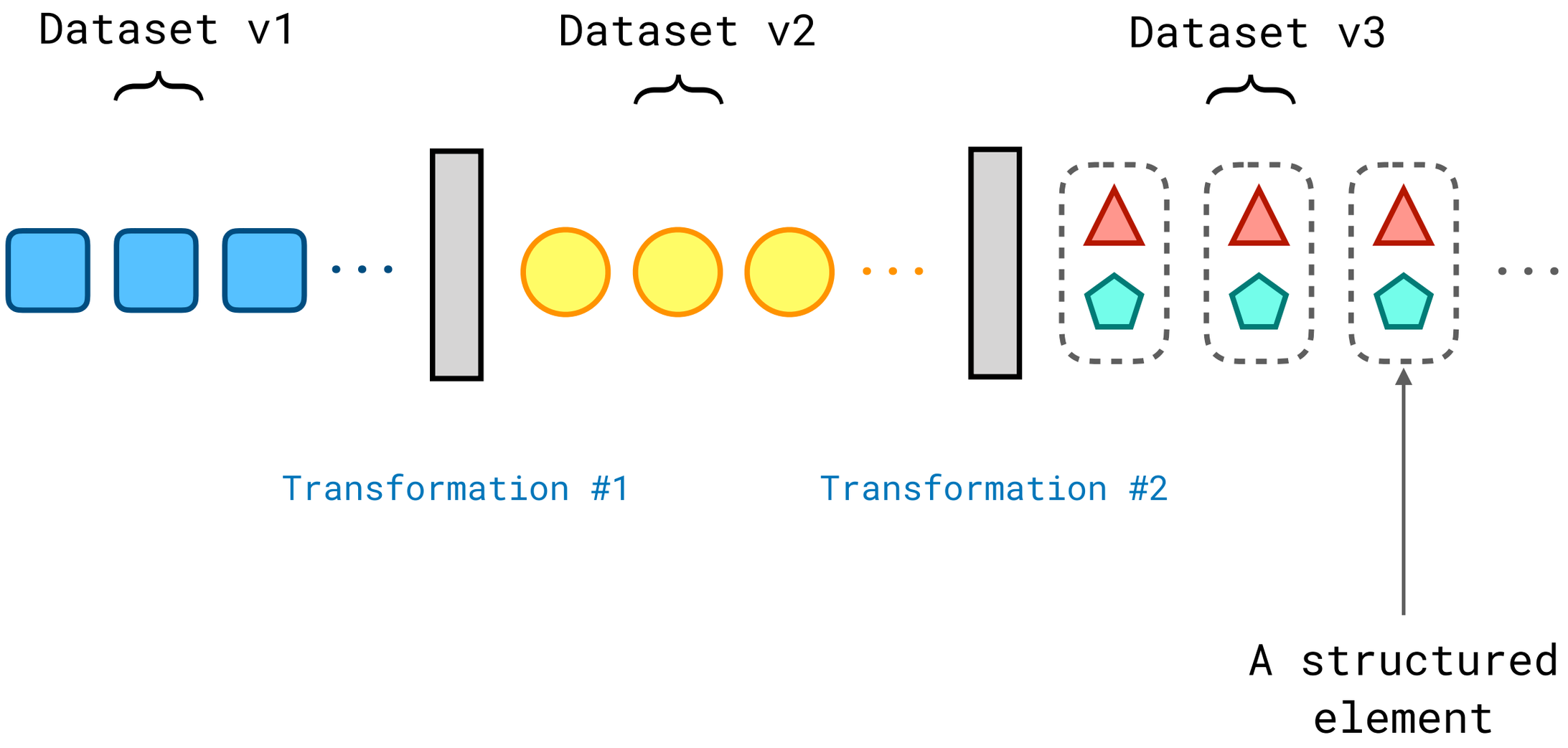
tf.data.Dataset Mechanism



tf.data.Dataset Mechanism



tf.data.Dataset Mechanism



Outline

Old Style vs. tf.data

tf.data.Dataset Mechanism

Using tf.data.Dataset

Common Transformations

tf.data motivation

Structured Elements

Data Sources

Outline

Old Style vs. tf.data

tf.data.Dataset Mechanism

Using tf.data.Dataset

Common Transformations

tf.data motivation

Structured Elements

Data Sources

Structured Elements

```
# tuple like
tf.data.Dataset.from_tensor_slices(
    ( [[1,2,3], [4,5,6]], [0, 1] )
)
```

Structured Elements

```
# tuple like
tf.data.Dataset.from_tensor_slices(
    ( [[1,2,3], [4,5,6]], [0, 1] )
)
(TensorSpec(shape=(3,), dtype=tf.int32),
 TensorSpec(shape=(), dtype=tf.int32))
```

Structured Elements

```
# tuple like
tf.data.Dataset.from_tensor_slices(
    ( [[1,2,3], [4,5,6]], [0, 1] )
)
(TensorSpec(shape=(3,), dtype=tf.int32),
 TensorSpec(shape=(), dtype=tf.int32))
```

```
# tuple of tuples
tf.data.Dataset.from_tensor_slices(
    ( ( [[1,2,3], [4,5,6]], ['a', 'b'] ), [0, 1] )
)
```

Structured Elements

```
# tuple like
tf.data.Dataset.from_tensor_slices(
    ( [[1,2,3], [4,5,6]], [0, 1] )
)
(TensorSpec(shape=(3,), dtype=tf.int
 TensorSpec(shape=(), dtype=tf.int32
```

```
def my_transform(element):
    inputs, targets = element
    ...
```

```
# tuple of tuples
tf.data.Dataset.from_tensor_slices(
    ( ( [[1,2,3], [4,5,6]], ['a', 'b'] ), [0, 1] )
)
((TensorSpec(shape=(3,), dtype=tf.int32),TensorSpec(shape=(), dtype=tf.string)),
  TensorSpec(shape=(), dtype=tf.int32))
```

Structured Elements

```
# tuple like
tf.data.Dataset.from_tensor_slices(
    ( [[1,2,3], [4,5,6]], [0, 1] )
)
(TensorSpec(shape=(3,), dtype=tf.int32),
 TensorSpec(shape=(), dtype=tf.int32))
```

```
# tuple of tuples
tf.data.Dataset.from_tensor_slices
    ( ( [[1,2,3], [4,5,6]], ['a',
)
((TensorSpec(shape=(3,), dtype=tf.int32),T
  TensorSpec(shape=(), dtype=tf.int32))
```

```
def my_transform(element):
    (num, chat), targets = element
    ...
```

Structured Elements

```
# dict like
tf.data.Dataset.from_tensor_slices(
    {'inputs': [[1,2,3], [4,5,6]], 'targets': [0, 1]}
)
```

Structured Elements

```
# dict like
tf.data.Dataset.from_tensor_slices(
    {‘inputs': [[1,2,3], [4,5,6]],
     'targets': [0, 1]}
)
{'inputs': TensorSpec(shape=(3,), dtype=tf.int32),
 'targets': TensorSpec(shape=(), dtype=tf.int32)}
```

Structured Elements

```
# dict like
tf.data.Dataset.from_tensor_slices(
    {'inputs': [[1,2,3], [4,5,6]],
     'targets': [0, 1]}
)
{'inputs': TensorSpec(shape=(3,), dtype=tf.int32),
 'targets': TensorSpec(shape=(), dtype=tf.int32)}
```

```
def my_transform(element):
    x = element['inputs']
    y = element['targets']
```

Structured Elements

```
# dict of dict
tf.data.Dataset.from_tensor_slices(
    {'inputs': {
        'num': [[1,2,3], [4,5,6]],
        'char': ['a', 'b']
    }, 'targets': [0, 1]}
)
```

Structured Elements

```
# dict of dict
tf.data.Dataset.from_tensor_sli
    {'inputs': {
        'num': [[1,2,3], [4,5,6
        'char': ['a', 'b']
    }, 'targets': [0, 1]}
)
```

```
def my_transform(element):
    n = element['inputs']['num']
    c = element['inputs']['char']
    y = element['targets']
```

```
{'inputs': {
    'num': TensorSpec(shape=(3,), dtype=tf.int32),
    'char': TensorSpec(shape=(), dtype=tf.string)
}, 'targets': TensorSpec(shape=(), dtype=tf.int32)}
```

What is the recommended way?

What is the recommended way?

It depends!

However, `(inputs, targets)` is common in TensorFlow.

What is the recommended way?

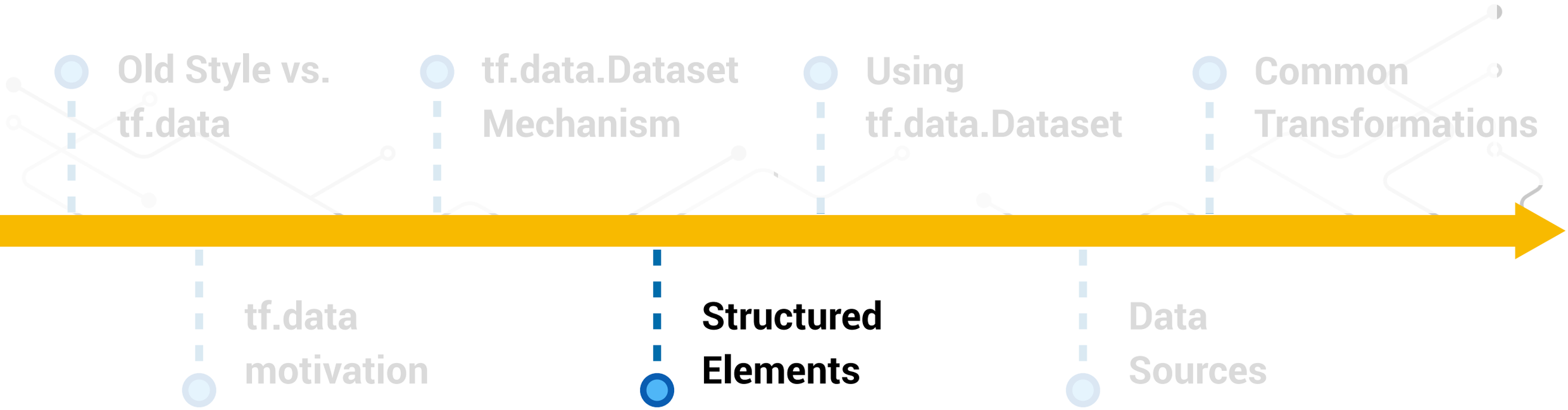
It depends!

However, `(inputs, targets)` is common in TensorFlow.

And, sometimes

`(inputs, targets, sample_weights)`

Outline


Old Style vs. tf.data
tf.data.Dataset Mechanism
Using tf.data.Dataset
Common Transformations
tf.data motivation
Structured Elements
Data Sources

Outline

- Old Style vs. tf.data
- tf.data.Dataset Mechanism
- **Using tf.data.Dataset**
- Common Transformations

- tf.data motivation
- Structured Elements
- Data Sources

But, how to use an instance of Dataset?

- Python `for`
- `tf.keras`

Using tf.data.Dataset: Python `for`

Using tf.data.Dataset: Python `for`

```
# A simple dataset, in which each element is a single integer
dataset = tf.data.Dataset.from_tensor_slices([8, 3, 0, 8, 2, 1])
```

Using tf.data.Dataset: Python `for`

```
# A simple dataset, in which each element is a single integer
dataset = tf.data.Dataset.from_tensor_slices([8, 3, 0, 8, 2, 1])

for elem in dataset:
    print(elem.numpy())
```

Using tf.data.Dataset: Python `for`

```
# A simple dataset, in which each element is a single integer
dataset = tf.data.Dataset.from_tensor_slices([8, 3, 0, 8, 2, 1])

for elem in dataset:
    print(elem.numpy())
```

```
8
3
0
8
2
1
```

Using tf.data.Dataset: Python `for` / A real-world usage

Using tf.data.Dataset: Python `for` / A real-world usage

```
def get_dataset():
    dataset = tf.data.Dataset.list_files(FILES)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

Using tf.data.Dataset: Python `for` / A real-world usage

```
def get_dataset():
    dataset = tf.data.Dataset.list_files(FILES)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset

def train_step(inputs, targets):
```

. . .

Using tf.data.Dataset: Python `for` / A real-world usage

```
def get_dataset():
    dataset = tf.data.Dataset.list_files(FILES)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset

def train_step(inputs, targets):
    ...

ds = get_dataset()
for inputs, targets in ds:
    train_step(inputs, targets)
```

Using tf.data.Dataset: `tf.keras`

Using tf.data.Dataset: `tf.keras`

```
def get_dataset(dir_path):
    # Create a dataset of (feature, label) pairs
    dataset = tf.data.Dataset.list_files(dir_path)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

Using tf.data.Dataset: `tf.keras`

```
def get_dataset(dir_path):
    # Create a dataset of (feature, label) pairs
    dataset = tf.data.Dataset.list_files(dir_path)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

```
model = get_model() # Sequential, Functional, or MSC
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Using tf.data.Dataset: `tf.keras`

```
def get_dataset(dir_path):
    # Create a dataset of (feature, label) pairs
    dataset = tf.data.Dataset.list_files(dir_path)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

```
model = get_model() # Sequential, Functional, or MSC
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])


train_ds = get_dataset('/path/to/train')
```

Using tf.data.Dataset: `tf.keras`

```
def get_dataset(dir_path):
    # Create a dataset of (feature, label) pairs
    dataset = tf.data.Dataset.list_files(dir_path)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

```
model = get_model() # Sequential, Functional, or MSC
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])


train_ds = get_dataset('/path/to/train')

# Train
model.fit(train_ds, epochs=10)
```

Using tf.data.Dataset: `tf.keras`

. . .

```python
train_ds = get_dataset('/path/to/train')

# Train
model.fit(train_ds, epochs=10)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')

# Train
model.fit(train_ds, epochs=10)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=10)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=10,
          validation_data=valid_ds)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=10,
          validation_data=valid_ds, validation_steps=100)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=10,
          validation_data=valid_ds, validation_steps=100)

# Test
test_ds = get_dataset('/path/to/test')
loss, accuracy = model.evaluate(test_ds)
```

Using tf.data.Dataset: `tf.keras`

```
...

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=10,
          validation_data=valid_ds, validation_steps=100)

# Test
test_ds = get_dataset('/path/to/test')
loss, accuracy = model.evaluate(test_ds)

# Predict
test_ds = get_dataset('/path/to/test')
result = model.predict(test_ds)
```

Using tf.data.Dataset: `tf.keras`

```
def get_dataset(dir_path):
    # Create a dataset of (feature, label) pairs
    dataset = tf.data.Dataset.list_files(dir_path)
    ...
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset

model = get_model() # Sequential, Functional, or MSC
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

train_ds = get_dataset('/path/to/train')
valid_ds = get_dataset('/path/to/valid')

# Train
model.fit(train_ds, epochs=2)
```

Outline

- Old Style vs. tf.data
- tf.data.Dataset Mechanism
- **Using tf.data.Dataset**
- Common Transformations

- tf.data motivation
- Structured Elements
- Data Sources

Outline

Old Style vs. tf.data

tf.data.Dataset Mechanism

Using tf.data.Dataset

Common Transformations

tf.data motivation

Structured Elements

Data Sources

Data sources: Reading the input data

- NumPy/Python arrays
- Python Generator
- Text Data
- CSV

Data Source: NumPy/Python arrays

Data Source: NumPy/Python arrays

```
train, test = tf.keras.datasets.mnist.load_data()

# images.shape = (60000, 28, 28)
# labels.shape = (60000,)
images, labels = train

dataset = tf.data.Dataset.from_tensor_slices((images, labels))
dataset
```

Data Source: NumPy/Python arrays

```
train, test = tf.keras.datasets.mnist.load_data()

# images.shape = (60000, 28, 28)
# labels.shape = (60000,)
images, labels = train

dataset = tf.data.Dataset.from_tensor_slices((images, labels))
dataset
```

```
<TensorSliceDataset shapes: ((28, 28), ()),
                    types: (tf.uint8, tf.uint8)>
```

Data Source: NumPy/Python arrays

```
...

for image, label in dataset:
    print(image.numpy().shape)
    print(label.numpy())
```

```
(28, 28)
5
...
```

Data Source: Python Generator

Data Source: Python Generator

```
# Generates [(0.0, 1), (1.0, 1), (2.0, 1), ...]
def count(stop):
    i = 0
    while i < stop:
        yield float(i), randint(0, 1)
        i += 1
```

Data Source: Python Generator

```
# Generates [(0.0, 1), (1.0, 1), (2.0, 1), ...]
def count(stop):
    i = 0
    while i < stop:
        yield float(i), randint(0, 1)
        i += 1

ds_counter = tf.data.Dataset.from_generator(
    count, args=[25],
    output_types=(?),
    output_shapes=(?)
)
```

Data Source: Python Generator

```
# Generates [(0.0, 1), (1.0, 1), (2.0, 1), ...]
def count(stop):
    i = 0
    while i < stop:
        yield float(i), randint(0, 1)
        i += 1

ds_counter = tf.data.Dataset.from_generator(
    count, args=[25],
    output_types=(tf.float32, tf.int32),
    output_shapes=((), ())
)
```

Data Source: Python Generator

```
# Generates [(0.0, 1), (1.0, 1), (2.0, 1), ...]
def count(stop):
    i = 0
    while i < stop:
        yield float(i), randint(0, 1)
        i += 1

ds_counter = tf.data.Dataset.from_generator(
    count, args=[25],
    output_types=(tf.float32, tf.int32),
    output_shapes=((), ())
)
ds_counter
```

```
<FlatMapDataset shapes: ((), ()), types: (tf.float32, tf.int32)>
```

Data Source: Python Generator

```
...

for element in ds_counter:
    print(element)


(<tf.Tensor: shape=(), dtype=float32, numpy=0.0>,
 <tf.Tensor: shape=(), dtype=int32, numpy=0>)
...
```

Data Source: Text Data

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt
```

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt
```

```
dataset = tf.data.TextLineDataset('dataset.txt')
dataset
```

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt

dataset = tf.data.TextLineDataset('dataset.txt')
dataset

<TextLineDatasetV2 shapes: (), types: tf.string>
```

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt
```

```
dataset = tf.data.TextLineDataset('dataset.txt')
dataset
```

```
<TextLineDatasetV2 shapes: (), types: tf.string>
```

```
dataset = dataset.map(tf.strings.split)
for line in dataset:
    print(line)
```

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt
```

```
dataset = tf.data.TextLineDataset('dataset.txt')
dataset
```

```
<TextLineDatasetV2 shapes: (), types: tf.string>
```

```
dataset = dataset.map(tf.strings.split)
for line in dataset:
    print(line)
```

```
tf.Tensor([b'Hello' b'World' b'!'], shape=(3,), dtype=string)
tf.Tensor([b'How' b'are' b'you' b'?'], shape=(4,), dtype=string)
```

Data Source: Text Data

```
$ echo “Hello World !” >> dataset.txt
$ echo “How are you ?” >> dataset.txt

dataset = tf.data.TextLineDataset([‘dataset_01.txt’, ...])
dataset

<TextLineDatasetV2 shapes: (), types: tf.string>
```

```
dataset = dataset.map(tf.strings.split)
for line in dataset:
    print(line)

tf.Tensor([b'Hello' b'World' b'!'], shape=(3,), dtype=string)
tf.Tensor([b'How' b'are' b'you' b'?'], shape=(4,), dtype=string)
```

Data Source: CSV

```
$ head “train.tsv”
```

```
survived,sex,age,n_siblings_spouses,parch,fare,class,deck,embark_town,alone
0,male,22.0,1,0,7.25,Third,unknown,Southampton,n
1,female,38.0,1,0,71.2833,First,C,Cherbourg,n
1,female,26.0,0,0,7.925,Third,unknown,Southampton,y
1,female,35.0,1,0,53.1,First,C,Southampton,n
0,male,28.0,0,0,8.4583,Third,unknown,Queenstown,y
0,male,2.0,3,1,21.075,Third,unknown,Southampton,n
1,female,27.0,0,2,11.1333,Third,unknown,Southampton,n
1,female,14.0,1,0,30.0708,Second,unknown,Cherbourg,n
1,female,4.0,1,1,16.7,Third,G,Southampton,n
```

Data Source: CSV

```
dataset = tf.data.experimental.make_csv_dataset(
    "train.csv", batch_size=4,
    label_name="survived",
    select_columns=['class', 'fare', 'survived']
)
```

Data Source: CSV

```
dataset = tf.data.experimental.make_csv_dataset(
    "train.csv", batch_size=4,
    label_name="survived",
    select_columns=['class', 'fare', 'survived']
)
```

```
(OrderedDict([('fare', TensorSpec(shape=(4,), dtype=tf.float32, name=None)),
              ('class', TensorSpec(shape=(4,), dtype=tf.string, name=None))]),
 TensorSpec(shape=(4,), dtype=tf.int32, name=None))
```

Data Source: CSV

```
...

for feature_batch, label_batch in dataset.take(1):
    print('class: ', feature_batch['class'].numpy())
    print('fare: ', feature_batch['fare'].numpy())
    print("'survived': ", label_batch.numpy())
```

Data Source: CSV

```
...

for feature_batch, label_batch in dataset.take(1):
    print('class: ', feature_batch['class'].numpy())
    print('fare: ', feature_batch['fare'].numpy())
    print("'survived': ", label_batch.numpy())
```

```
class:  [b'Second' b'Second' b'Second' b'First']
fare:  [ 10.5    12.35   13.    135.6333]
'survived':  [0 1 1 1]
```

Outline

Old Style vs. tf.data

tf.data.Dataset Mechanism

Using tf.data.Dataset

Common Transformations

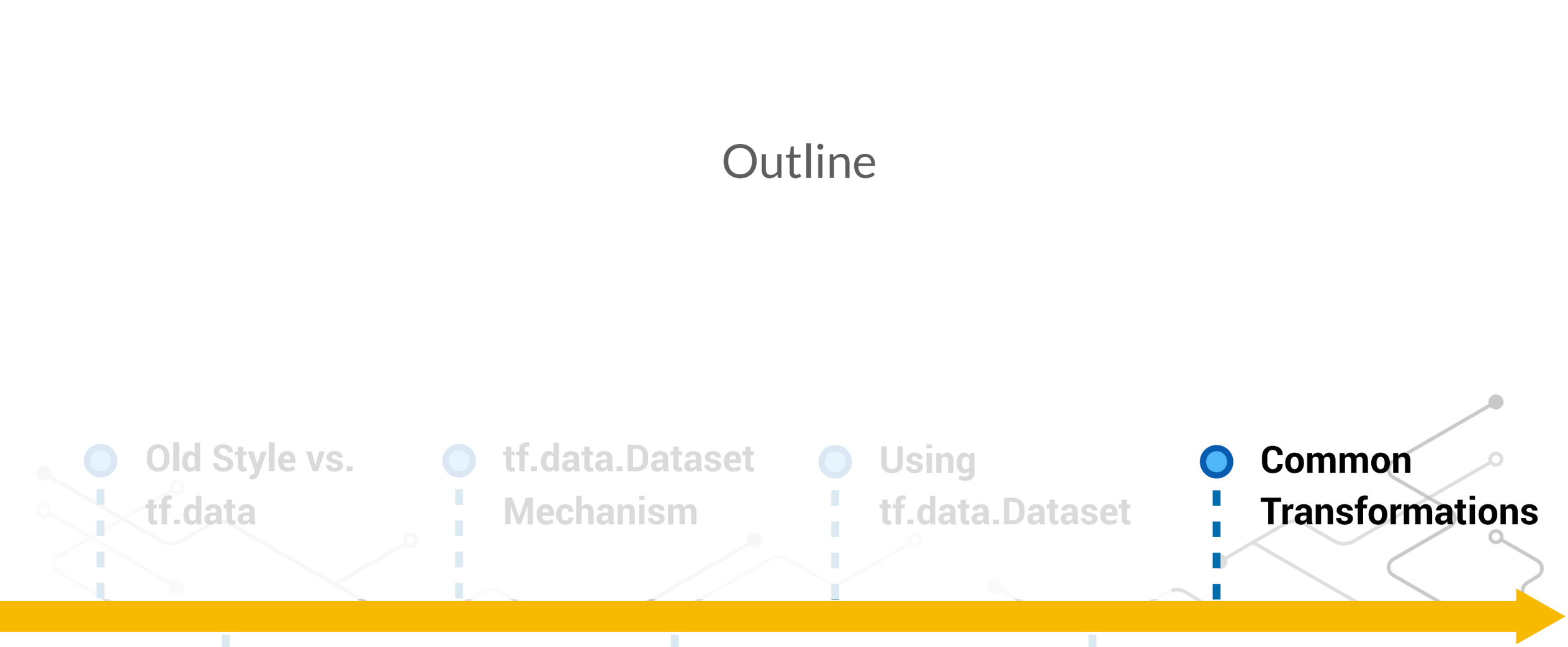
tf.data motivation

Structured Elements

Data Sources

Outline

- Old Style vs. tf.data
- tf.data.Dataset Mechanism
- Using tf.data.Dataset
- **Common Transformations**



- tf.data motivation
- Structured Elements
- Data Sources

Common Transformations

- `.map(...)`
- `.flat_map(...)`
- `.filter(...)`
- `.reduce(...)`
- `.skip(...)`
- `Dataset.zip(...)`

Thank you!