Google Cloud

TensorFlow Transform

Lak Lakshmanan

Learn how to...

Learn how to...

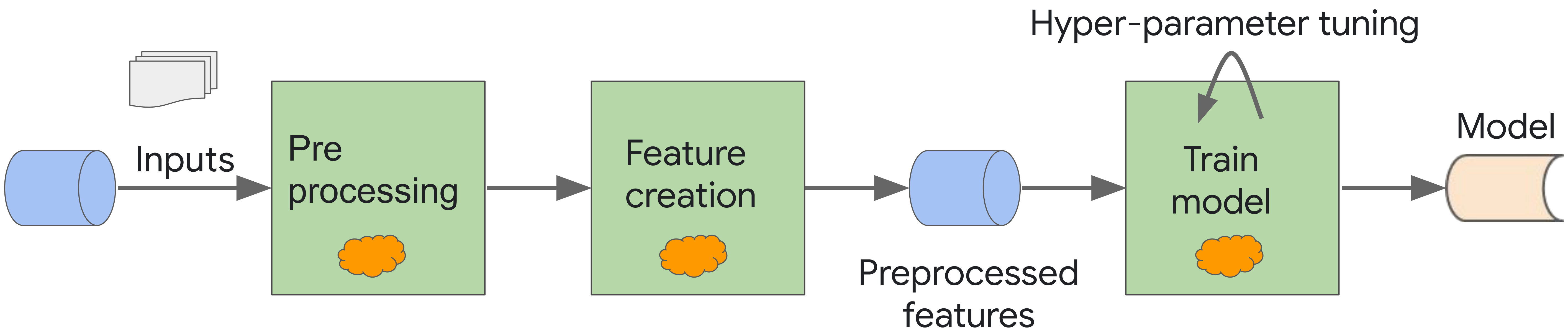
Implement feature preprocessing and feature creation using tf.transform

Learn how to...

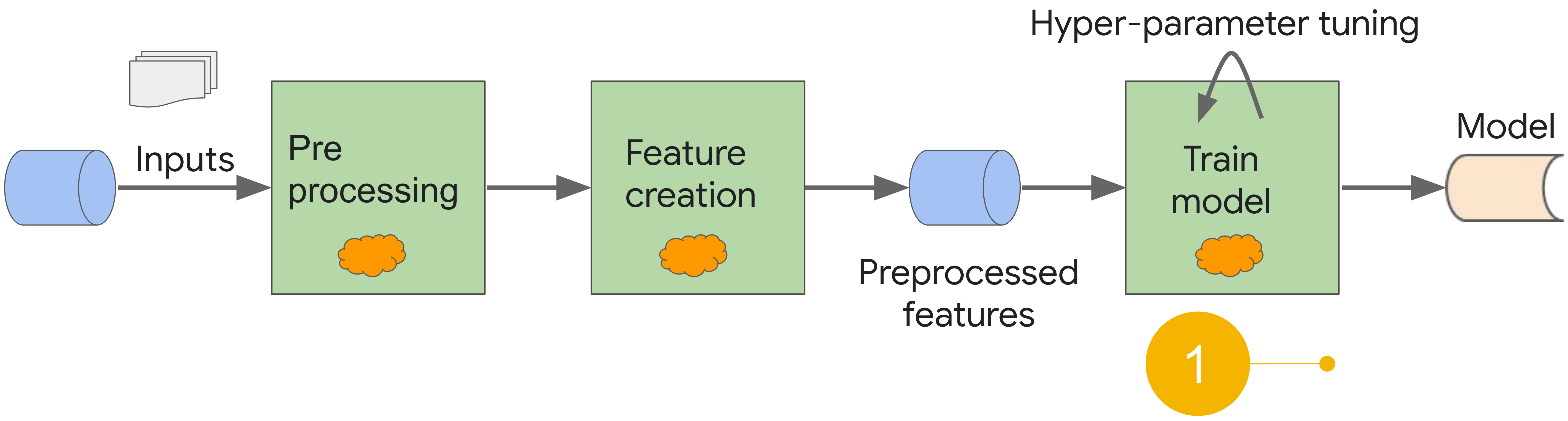
Implement feature preprocessing and feature creation using tf.transform

Carry out feature processing efficiently, at scale and on streaming data

Recall that there are three possible places to do feature engineering, each of which has its pros and cons



Recall that there are three possible places to do feature engineering, each of which has its pros and cons



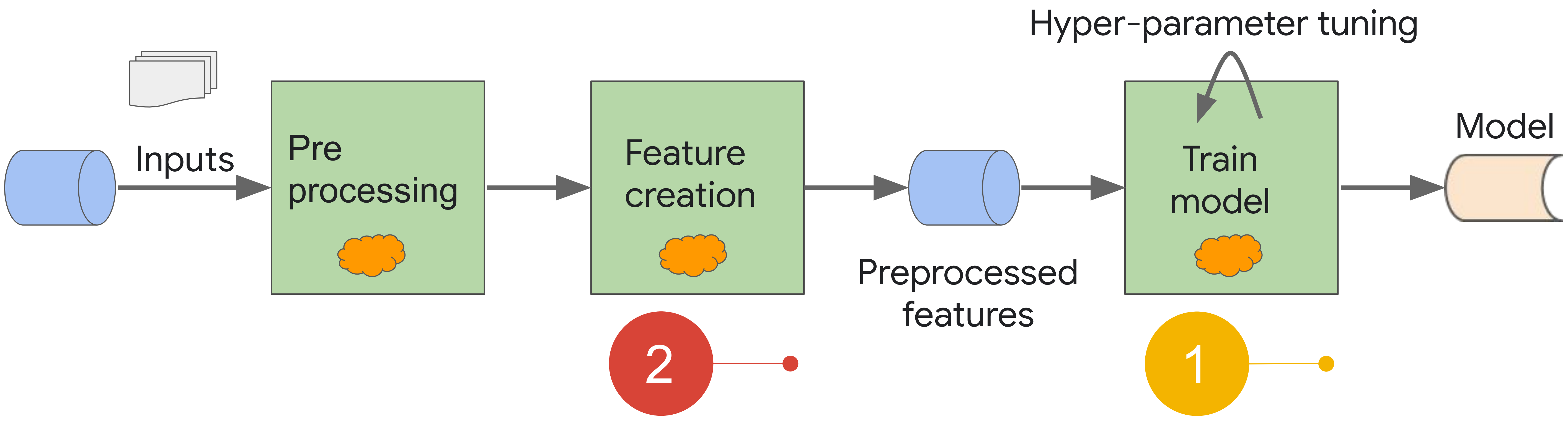
TensorFlow*

*efficient

*tf methods only

*this input only

Recall that there are three possible places to do feature engineering, each of which has its pros and cons



Beam*

*in a pipeline

*Python/Java code

*time-windows

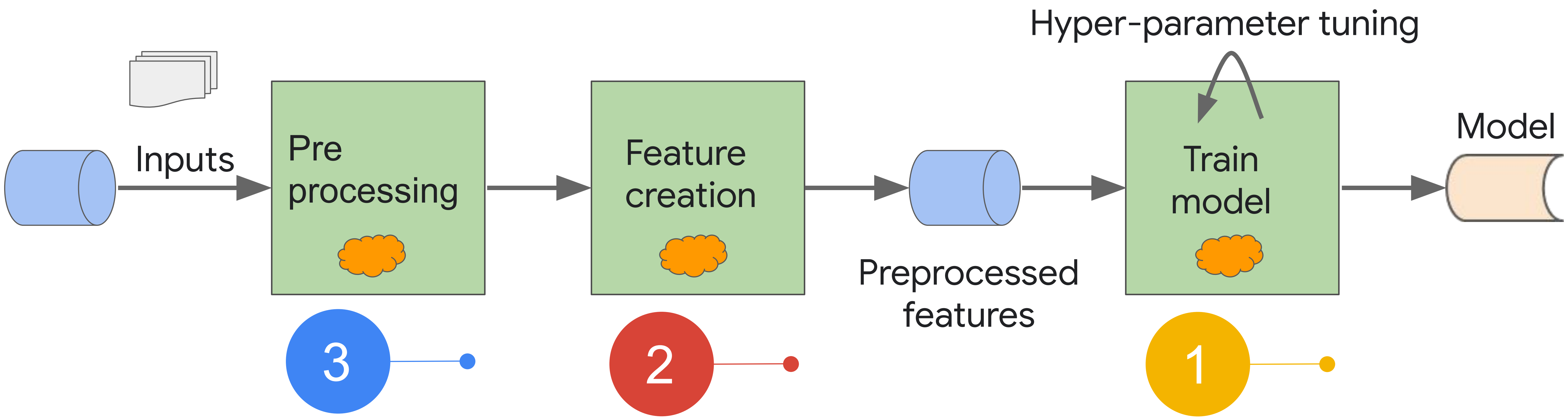
TensorFlow*

*efficient

*tf methods only

*this input only

Recall that there are three possible places to do feature engineering, each of which has its pros and cons

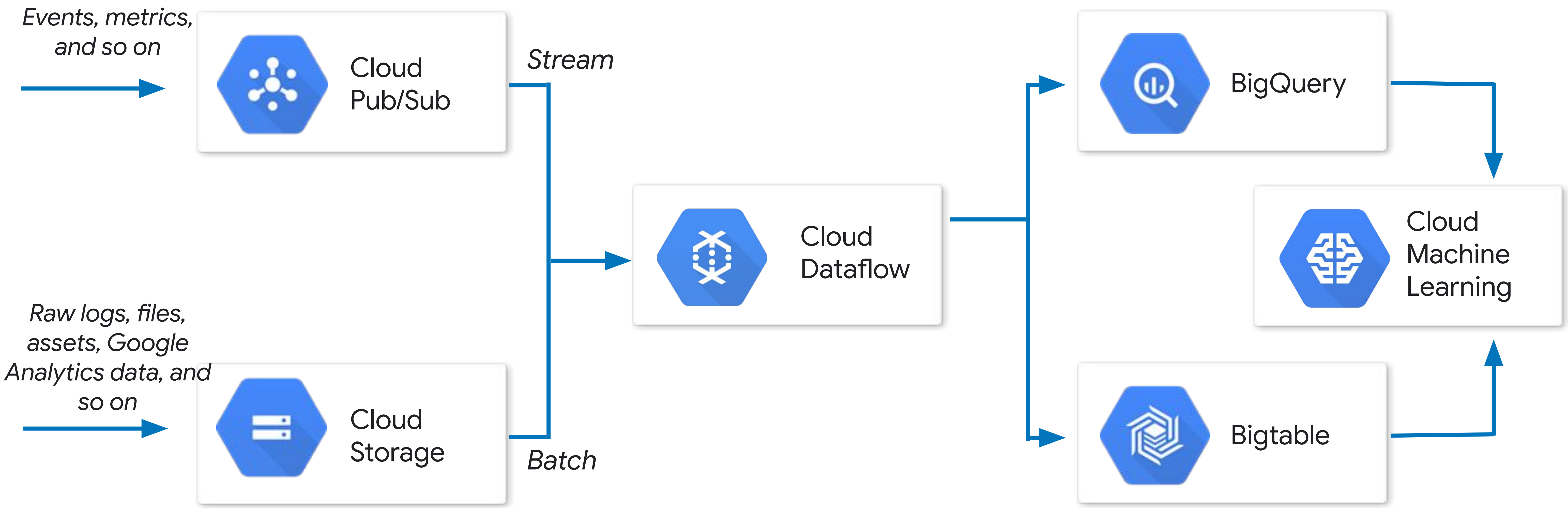


tf.transform*
*efficient
*tf methods only
*aggregates

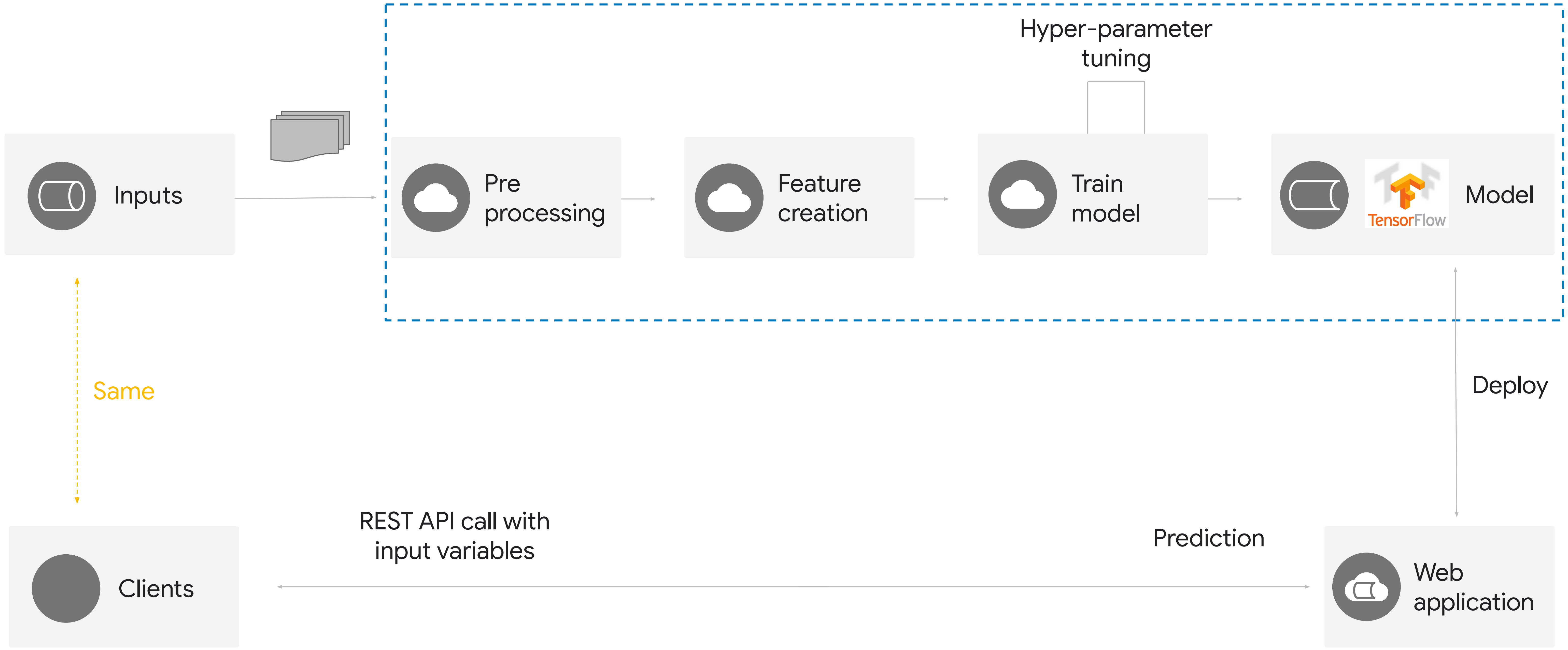
Beam*
*in a pipeline
*Python/Java code
*time-windows

TensorFlow*
*efficient
*tf methods only
*this input only

Dataflow preprocessing works in the context of a pipeline



TensorFlow is good for on-demand, on-the-fly processing



tf.transform is a hybrid of
Beam and TensorFlow

tf.transform is a hybrid of Beam and TensorFlow

Find min/max value of a numeric feature

tf.transform is a hybrid of Beam and TensorFlow

Find min/max value of a numeric feature

Scale inputs by the min & max

tf.transform is a hybrid of Beam and TensorFlow

Find min/max value of a numeric feature

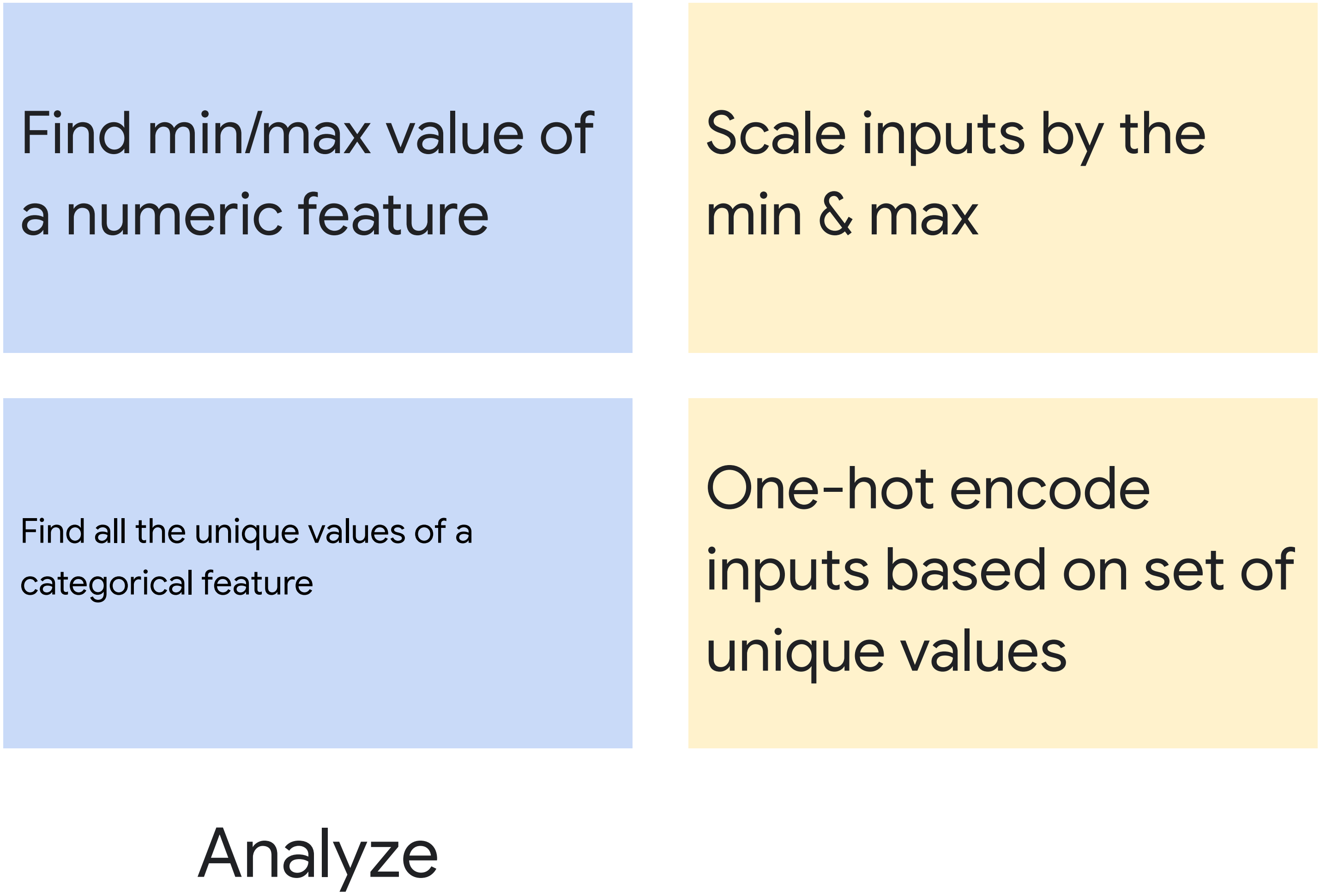
Scale inputs by the min & max

Find all the unique values of a categorical feature

tf.transform is a hybrid of Beam and TensorFlow

Find min/max value of a numeric feature	Scale inputs by the min & max
Find all the unique values of a categorical feature	One-hot encode inputs based on set of unique values

tf.transform is a hybrid of Beam and TensorFlow



Find min/max value of a numeric feature

Scale inputs by the min & max

Find all the unique values of a categorical feature

One-hot encode inputs based on set of unique values

Analyze

tf.transform is a hybrid of Beam and TensorFlow

Analyze	Transform
Find min/max value of a numeric feature	Scale inputs by the min & max
Find all the unique values of a categorical feature	One-hot encode inputs based on set of unique values

tf.transform is a hybrid of Beam and TensorFlow

Find min/max value of a numeric feature	Scale inputs by the min & max
Find all the unique values of a categorical feature	One-hot encode inputs based on set of unique values
Analyze	Transform
Beam	TensorFlow

tf.transform provides two PTransforms

tf.transform provides two PTransforms

`AnalyzeAndTransformDataset`

tf.transform provides two PTransforms

`AnalyzeAndTransformDataset`

Executed in Beam to create the training dataset

tf.transform provides two PTransforms

`AnalyzeAndTransformDataset`

Executed in Beam to create the training dataset

`TransformDataset`

tf.transform provides two PTransforms

`AnalyzeAndTransformDataset`

Executed in Beam to create the training dataset

`TransformDataset`

Executed in Beam to create the evaluation dataset

tf.transform provides two PTransforms

`AnalyzeAndTransformDataset`

Executed in Beam to create the training dataset

`TransformDataset`

Executed in Beam to create the evaluation dataset

The underlying transformations are executed in TensorFlow at prediction time

tf.transform has two phases

tf.transform has two phases

Analysis phase (compute min/max/vocab etc. using Beam)

tf.transform has two phases

Analysis phase (compute min/max/vocab etc. using Beam)

- Executed in Beam while creating training dataset

tf.transform has two phases

Analysis phase (compute min/max/vocab etc. using Beam)

- Executed in Beam while creating training dataset

Transform phase (scale/vocabulary etc. using TensorFlow)

tf.transform has two phases

Analysis phase (compute min/max/vocab etc. using Beam)

- Executed in Beam while creating training dataset

Transform phase (scale/vocabulary etc. using TensorFlow)

- Executed in TensorFlow during prediction

tf.transform has two phases

Analysis phase (compute min/max/vocab etc. using Beam)

- Executed in Beam while creating training dataset

Transform phase (scale/vocabulary etc. using TensorFlow)

- Executed in TensorFlow during prediction

- Executed in Beam to create training/evaluation datasets

First, set up the schema of the training dataset

First, set up the schema of the training dataset

```
raw_data_schema = {
    colname : dataset_schema.ColumnSchema(tf.string, ...)
        for colname in 'dayofweek,key'.split(',')
}
```

First, set up the schema of the training dataset

```
raw_data_schema = {
    colname : dataset_schema.ColumnSchema(tf.string, ...)
        for colname in 'dayofweek,key'.split(',')
}
```

TensorFlow type for input column

First, set up the schema of the training dataset

```
raw_data_schema = {
    colname : dataset_schema.ColumnSchema(tf.string, ...)
        for colname in 'dayofweek,key'.split(',')
}
raw_data_schema.update({
      colname : dataset_schema.ColumnSchema(tf.float32, ...)
        for colname in 'fare_amount,pickuplon, … ,dropofflat'.split(',')
})
```

TensorFlow type for input column

float32

First, set up the schema of the training dataset

TensorFlow type for input column

```
raw_data_schema = {
    colname : dataset_schema.ColumnSchema(tf.string, ...)
        for colname in 'dayofweek,key'.split(',')
}
raw_data_schema.update({
      colname : dataset_schema.ColumnSchema(tf.float32, ...)
        for colname in 'fare_amount,pickuplon, … ,dropofflat'.split(',')
})
raw_data_metadata =
    dataset_metadata.DatasetMetadata(dataset_schema.Schema(raw_data_schema))
```

float32

First, set up the schema of the training dataset

```
raw_data_schema = {                       TensorFlow type for input column
    colname : dataset_schema.ColumnSchema(tf.string, ...)
        for colname in 'dayofweek,key'.split(',')
}
raw_data_schema.update({                                    float32
      colname : dataset_schema.ColumnSchema(tf.float32, ...)
        for colname in 'fare_amount,pickuplon, … ,dropofflat'.split(',')
})
raw_data_metadata =   Use the schema to create metadata “template”
    dataset_metadata.DatasetMetadata(dataset_schema.Schema(raw_data_schema))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
     | beam.io.Read(beam.io.BigQuerySource(query=myquery, use_standard_sql=True))
     | beam.Filter(is_valid))


transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
      | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

1. Read in data as usual for Beam

```
raw_data = (p
    | beam.io.Read(beam.io.BigQuerySource(query=myquery, use_standard_sql=True))
    | beam.Filter(is_valid))


transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
     | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

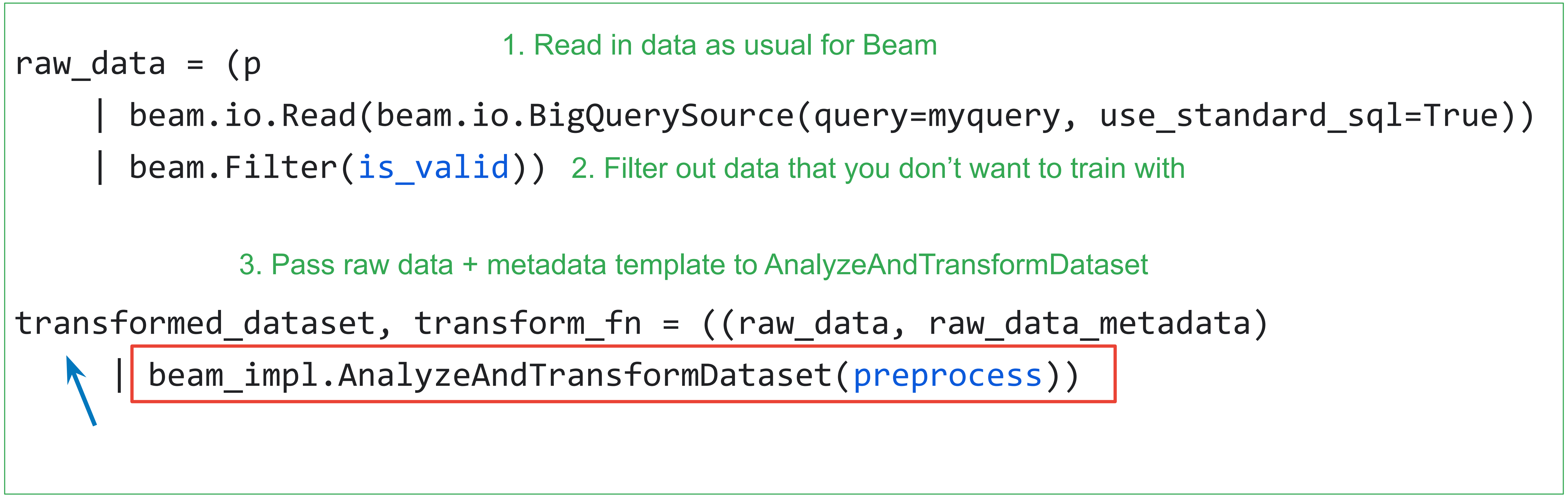
1. Read in data as usual for Beam

```
raw_data = (p
     | beam.io.Read(beam.io.BigQuerySource(query=myquery, use_standard_sql=True))
     | beam.Filter(is_valid))
```

2. Filter out data that you don’t want to train with

```
transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
      | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function



1. Read in data as usual for Beam

```
raw_data = (p
    | beam.io.Read(beam.io.BigQuerySource(query=myquery, use_standard_sql=True))
    | beam.Filter(is_valid))
```

2. Filter out data that you don’t want to train with

3. Pass raw data + metadata template to AnalyzeAndTransformDataset

```
transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
    | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

1. Read in data as usual for Beam

```
raw_data = (p
    | beam.io.Read(beam.io.BigQuerySource(query=myquery, use_standard_sql=True))
    | beam.Filter(is_valid))
```

2. Filter out data that you don’t want to train with

3. Pass raw data + metadata template to AnalyzeAndTransformDataset

```
transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
      | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

4. Get back transformed dataset and a reusable transform function

Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data |
   tfrecordio.WriteToTFRecord(
     os.path.join(OUTPUT_DIR,'train'),

     coder=ExampleProtoCoder(
            transformed_metadata.schema)
   )
```

Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data |
   tfrecordio.WriteToTFRecord(
     os.path.join(OUTPUT_DIR,'train'),
                  The filenames will be like train-0003-of0015
     coder=ExampleProtoCoder(
            transformed_metadata.schema)
   )
```

Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data |
   tfrecordio.WriteToTFRecord(
     os.path.join(OUTPUT_DIR,'train'),
```

The filenames will be like train-0003-of0015

```
     coder=ExampleProtoCoder(
            transformed_metadata.schema)
   )
```

Note that we use the transformed metadata schema here

The preprocessing function is restricted to TensorFlow functions

```
transformed_dataset, transform_fn =
((raw_data, raw_data_metadata)
     |
beam_impl.AnalyzeAndTransformDataset(
                  preprocess))
```

The preprocessing function is restricted to TensorFlow functions

```
transformed_dataset, transform_fn =
((raw_data, raw_data_metadata)
     |
beam_impl.AnalyzeAndTransformDataset(
                  preprocess))
```

The things you do in preprocess() will get added to the TensorFlow graph, and be executed in TensorFlow during serving

The preprocessing function is restricted to functions you can call from TensorFlow graph

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):
```

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

   result = {}
```

Create features from the input tensors and put into “results” dict

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

  result = {}   Create features from the input tensors and put into “results” dict

  result['fare_amount'] = inputs['fare_amount'] Pass through
```

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

   result = {}   Create features from the input tensors and put into “results” dict

   result['fare_amount'] = inputs['fare_amount'] Pass through

   result['dayofweek'] = tft.string_to_int(inputs['dayofweek']) vocabulary
```

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

   result = {}   Create features from the input tensors and put into “results” dict

   result['fare_amount'] = inputs['fare_amount'] Pass through

   result['dayofweek'] = tft.string_to_int(inputs['dayofweek']) vocabulary
   ...
   result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat'])) scaling
```

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

   result = {}   Create features from the input tensors and put into “results” dict

   result['fare_amount'] = inputs['fare_amount'] Pass through

   result['dayofweek'] = tft.string_to_int(inputs['dayofweek']) vocabulary
   ...
   result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat'])) scaling

   result['passengers'] = tf.cast(inputs['passengers'], tf.float32) Other TF fns
```

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):

   result = {}   Create features from the input tensors and put into “results” dict

   result['fare_amount'] = inputs['fare_amount'] Pass through

   result['dayofweek'] = tft.string_to_int(inputs['dayofweek']) vocabulary
   ...
   result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat'])) scaling

   result['passengers'] = tf.cast(inputs['passengers'], tf.float32) Other TF fns

   return result
```

Analyze and Transform happens on the training dataset

```
transformed_dataset, transform_fn =
((raw_data, raw_data_metadata)
     |
beam_impl.AnalyzeAndTransformDataset(
                  preprocess))
```

Writing out the eval dataset is similar, except that we reuse the transform function computed from the training data

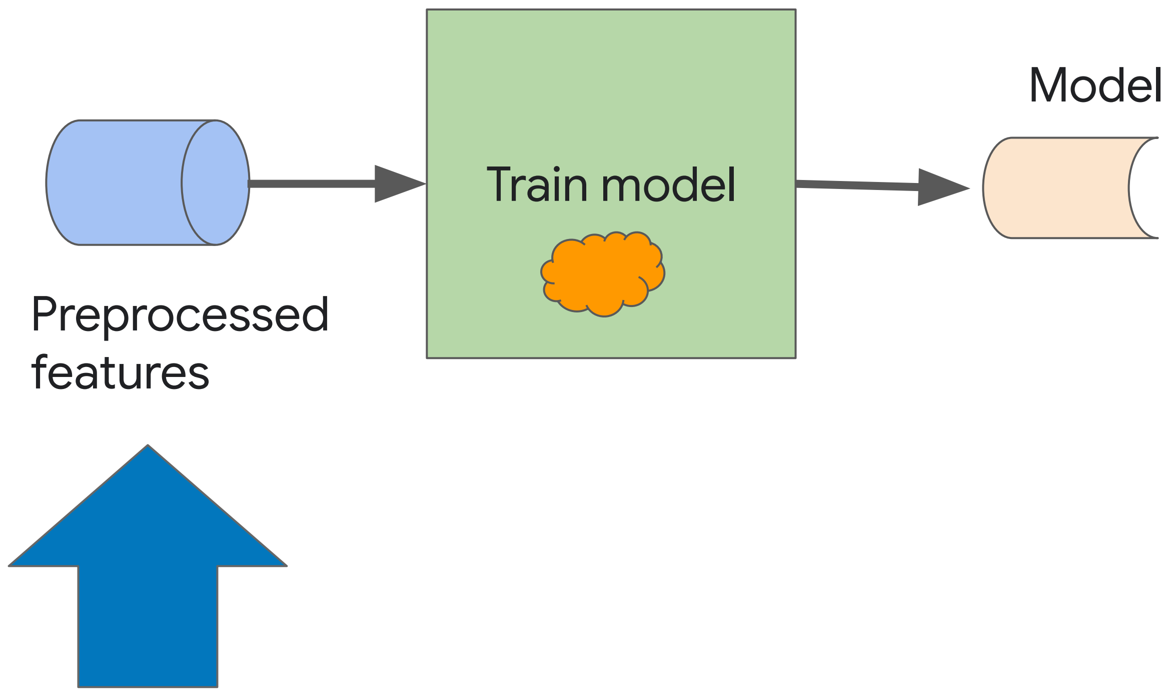
```
raw_test_data = (p
        | beam.io.Read(beam.io.BigQuerySource(...)
        | 'eval_filter' >> beam.Filter(is_valid))
transformed_test_dataset = (((raw_test_data, raw_data_metadata), transform_fn)
        | beam_impl.TransformDataset())
```

Writing out the eval dataset is similar, except that we reuse the transform function computed from the training data

```
raw_test_data = (p
        | beam.io.Read(beam.io.BigQuerySource(...)
        | 'eval_filter' >> beam.Filter(is_valid))
transformed_test_dataset = (((raw_test_data, raw_data_metadata), transform_fn)
        | beam_impl.TransformDataset())

transformed_test_data, _ = transformed_test_dataset
_ = transformed_test_data | tfrecordio.WriteToTFRecord(
          os.path.join(OUTPUT_DIR, 'eval'),
          coder=example_proto_coder.ExampleProtoCoder(
              transformed_metadata.schema))
```

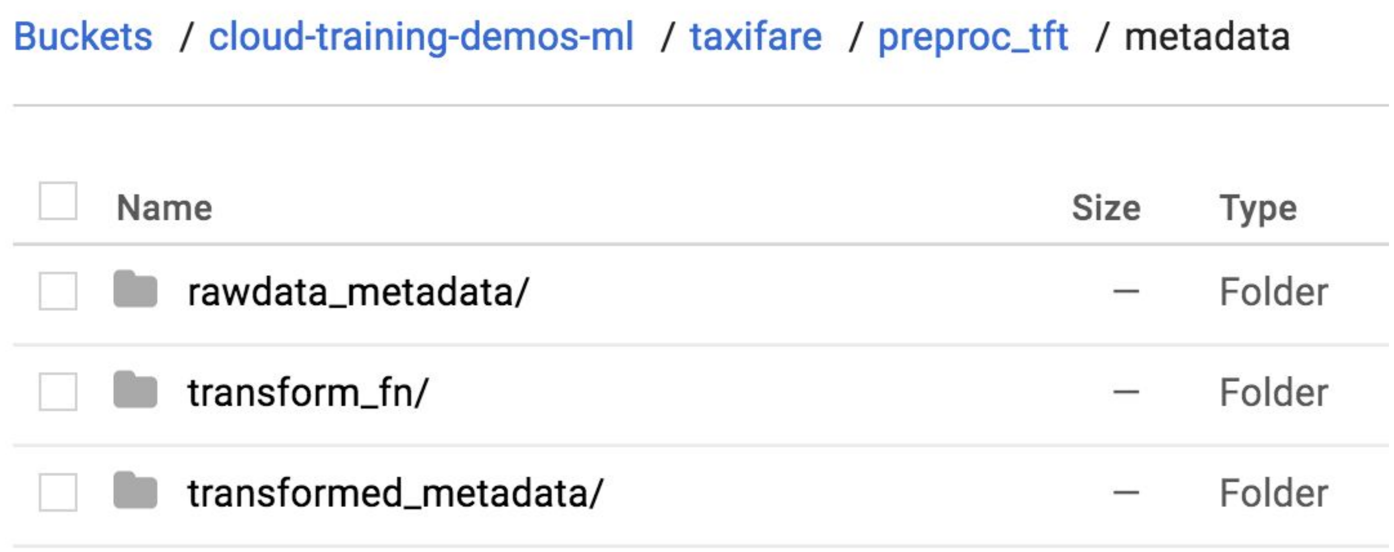
For training and evaluation, we created preprocessed features using Beam



Created by
AnalyzeAndTransformDataset
Or by TransformDataset

For serving, we need to write out the transformation metadata

```
_ = transform_fn |
  transform_fn_io.WriteTransformFn(
     os.path.join(OUTPUT_DIR,
'metadata')))
```



Buckets / cloud-training-demos-ml / taxifare / preproc_tft / metadata

□	Name	Size	Type
□	rawdata_metadata/	–	Folder
□	transform_fn/	–	Folder
□	transformed_metadata/	–	Folder

Change input function to read preprocessed features

```
def read_dataset(args, mode):
   if mode == tf.estimator.ModeKeys.TRAIN:
       input_paths = args['train_data_paths']
   else:
       input_paths = args['eval_data_paths']
   transformed_metadata = metadata_io.read_metadata(
             os.path.join(args['metadata_path'], 'transformed_metadata'))
   return input_fn_maker.build_training_input_fn(
         metadata = transformed_metadata,
         file_pattern = (
             input_paths[0] if len(input_paths) == 1 else input_paths),
        ...)
```

Reading transformed features

Change input function to read preprocessed features

```
def read_dataset(args, mode):
   if mode == tf.estimator.ModeKeys.TRAIN:
       input_paths = args['train_data_paths']
   else:
       input_paths = args['eval_data_paths']
   transformed_metadata = metadata_io.read_metadata(
             os.path.join(args['metadata_path'], 'transformed_metadata'))
   return input_fn_maker.build_training_input_fn(
         metadata = transformed_metadata,
         file_pattern = (
             input_paths[0] if len(input_paths) == 1 else input_paths),
        ...)
```

Reading transformed features

Change input function to read preprocessed features

```
def read_dataset(args, mode):
   if mode == tf.estimator.ModeKeys.TRAIN:
       input_paths = args['train_data_paths']
   else:
       input_paths = args['eval_data_paths']
   transformed_metadata = metadata_io.read_metadata(
             os.path.join(args['metadata_path'], 'transformed_metadata'))
   return input_fn_maker.build_training_input_fn(
         metadata = transformed_metadata,
         file_pattern = (
             input_paths[0] if len(input_paths) == 1 else input_paths),
        ...)
```

Reading transformed features

The serving input function accepts the raw data

```
def make_serving_input_fn(args):
   raw_metadata = metadata_io.read_metadata(
       os.path.join(args['metadata_path'], 'rawdata_metadata'))
   transform_savedmodel_dir = (
       os.path.join(args['metadata_path'], 'transform_fn'))
   return input_fn_maker.build_parsing_transforming_serving_input_receiver_fn(
     raw_metadata,
     transform_savedmodel_dir,
     exclude_raw_keys = [LABEL_COLUMN])
```

The serving input function accepts the raw data

```
def make_serving_input_fn(args):
   raw_metadata = metadata_io.read_metadata(
       os.path.join(args['metadata_path'], 'rawdata_metadata'))
   transform_savedmodel_dir = (
       os.path.join(args['metadata_path'], 'transform_fn'))
   return input_fn_maker.build_parsing_transforming_serving_input_receiver_fn(
     raw_metadata,
     transform_savedmodel_dir,
     exclude_raw_keys = [LABEL_COLUMN])
```

The serving input function accepts the raw data

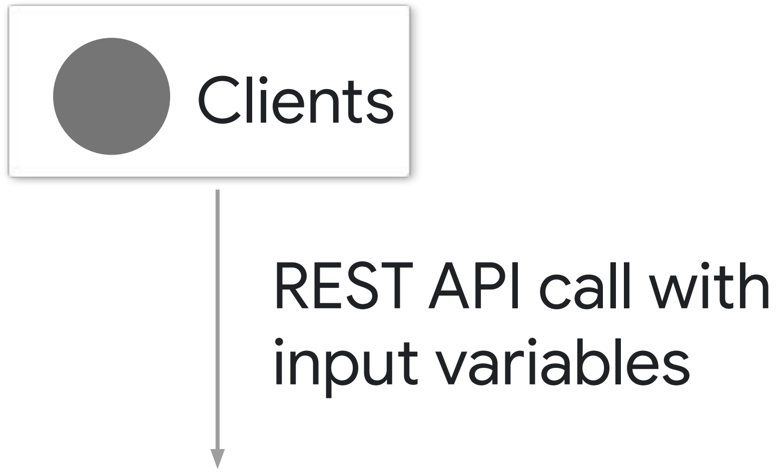
```
def make_serving_input_fn(args):
   raw_metadata = metadata_io.read_metadata(
       os.path.join(args['metadata_path'], 'rawdata_metadata'))
   transform_savedmodel_dir = (
       os.path.join(args['metadata_path'], 'transform_fn'))
   return input_fn_maker.build_parsing_transforming_serving_input_receiver_fn(
     raw_metadata,
     transform_savedmodel_dir,
     exclude_raw_keys = [LABEL_COLUMN])
```

The serving input function accepts the raw data

```
def make_serving_input_fn(args):
   raw_metadata = metadata_io.read_metadata(
       os.path.join(args['metadata_path'], 'rawdata_metadata'))
   transform_savedmodel_dir = (
       os.path.join(args['metadata_path'], 'transform_fn'))
   return input_fn_maker.build_parsing_transforming_serving_input_receiver_fn(
     raw_metadata,
     transform_savedmodel_dir,
     exclude_raw_keys = [LABEL_COLUMN])
```

The model graph includes the preprocessing code

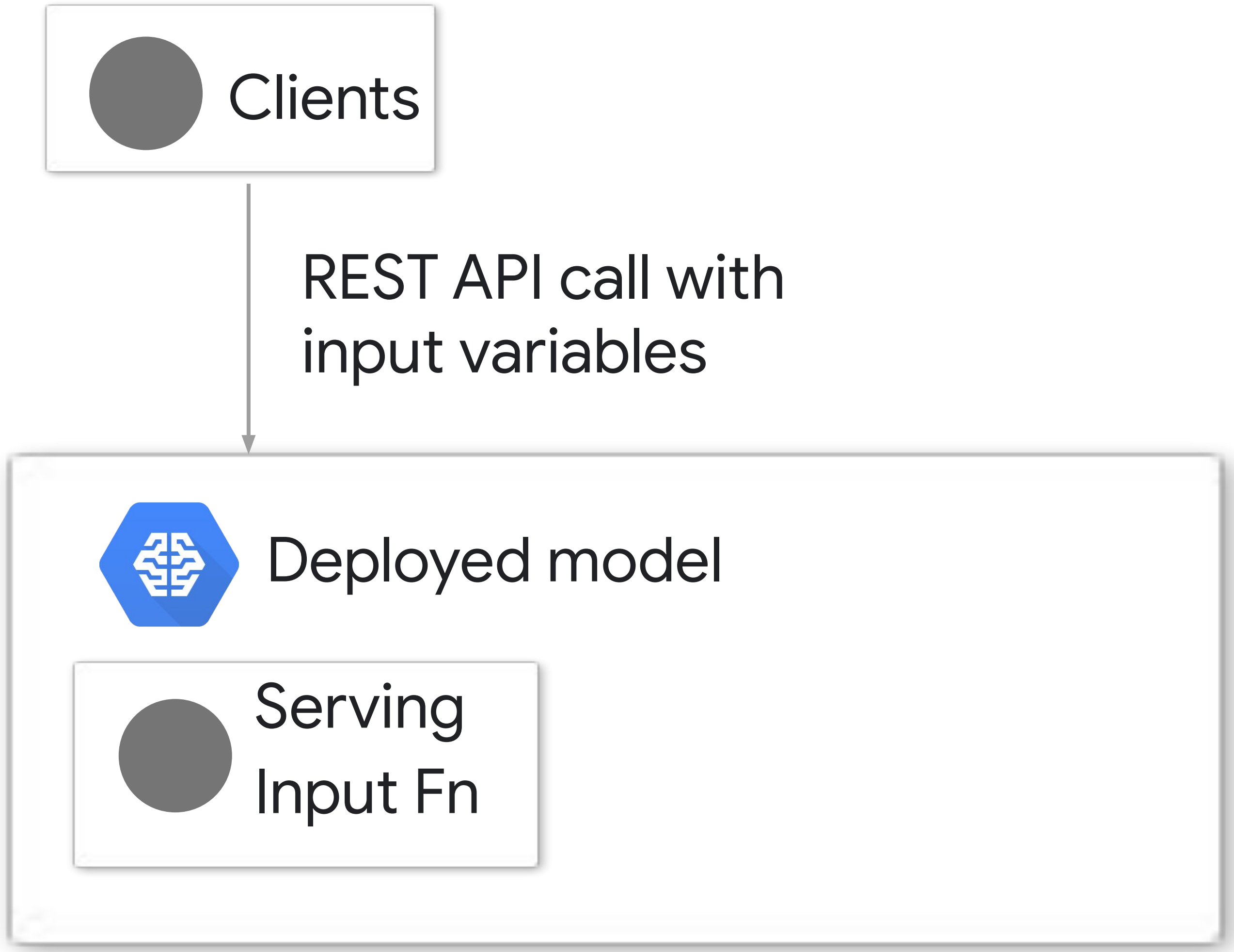
The model graph includes the preprocessing code



Clients

REST API call with input variables

The model graph includes the preprocessing code



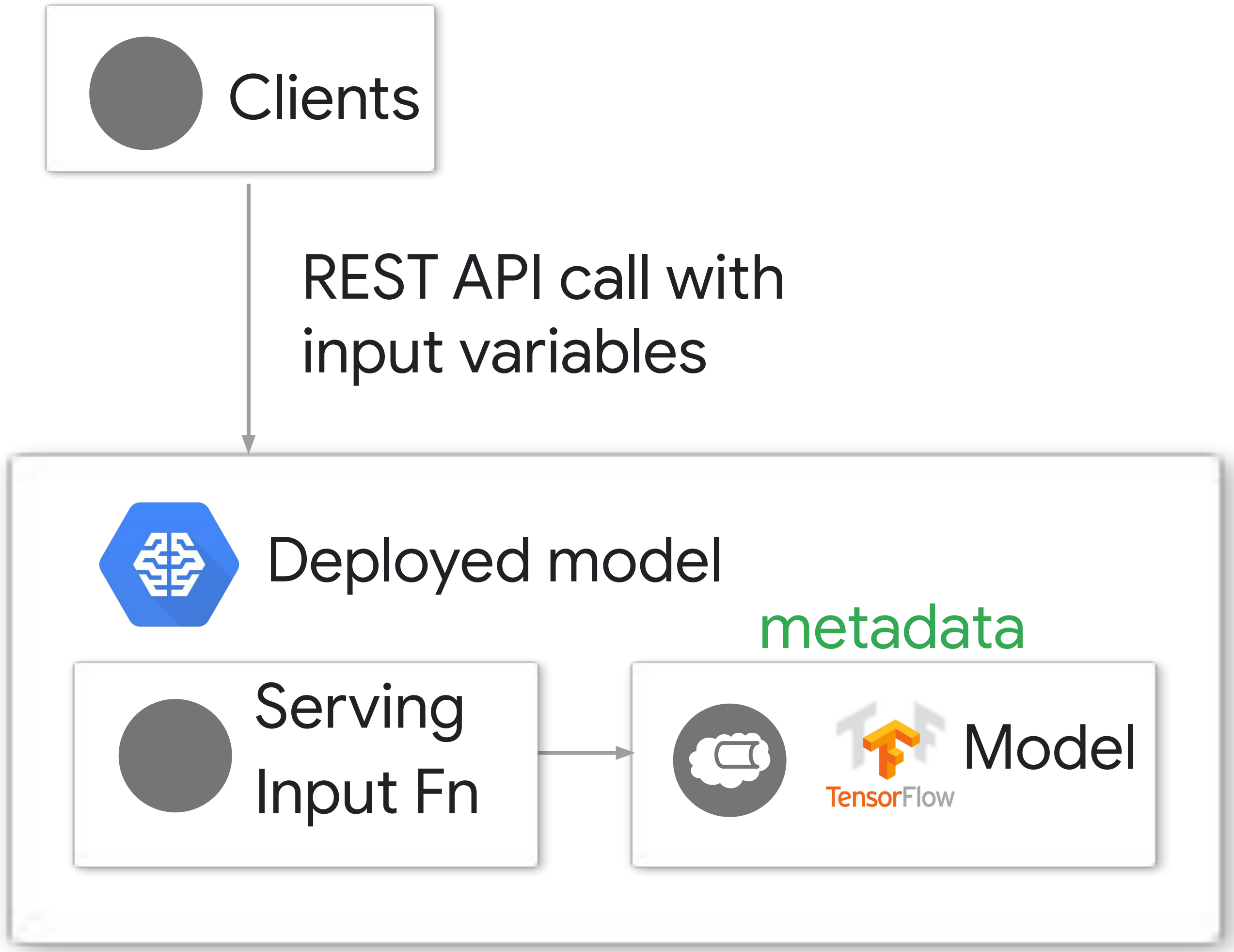
Clients

REST API call with input variables

Deployed model

Serving Input Fn

The model graph includes the preprocessing code



Clients

REST API call with input variables

Deployed model

metadata

Serving Input Fn

Model

Lab

Exploring tf.transform

Look at tftransform.ipynb

cloud.google.com