



# Introduction to TFX pipelines

Doug Kelly  
ML Solutions Engineer, Google Cloud

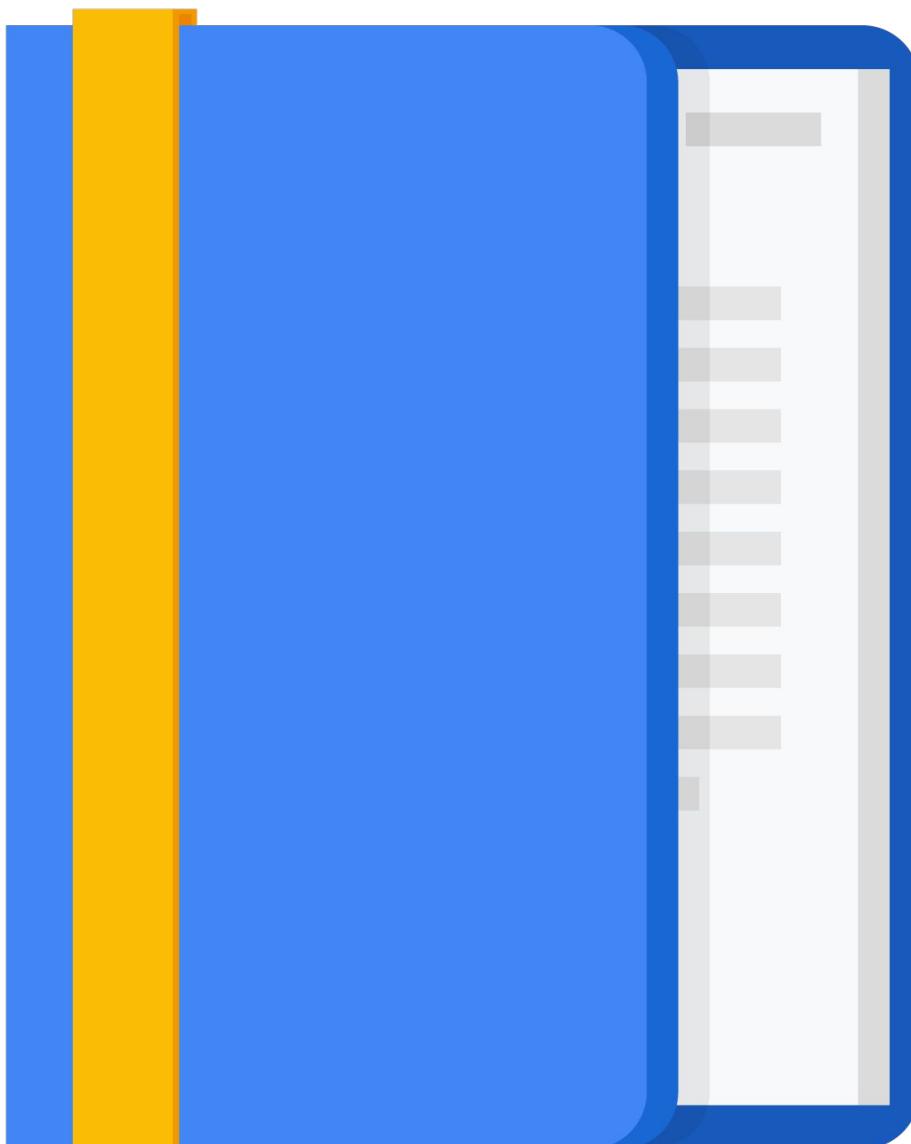
---

# Agenda

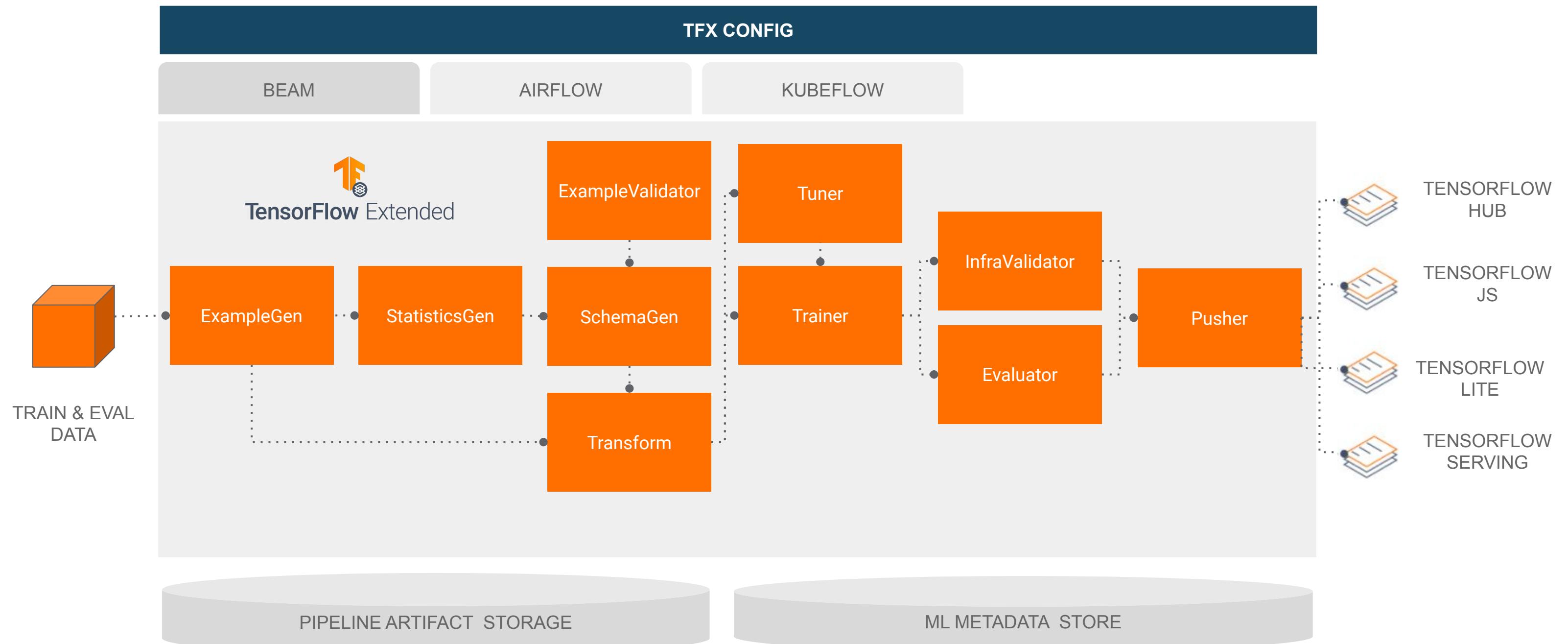
TensorFlow Extended (TFX)

TFX standard components

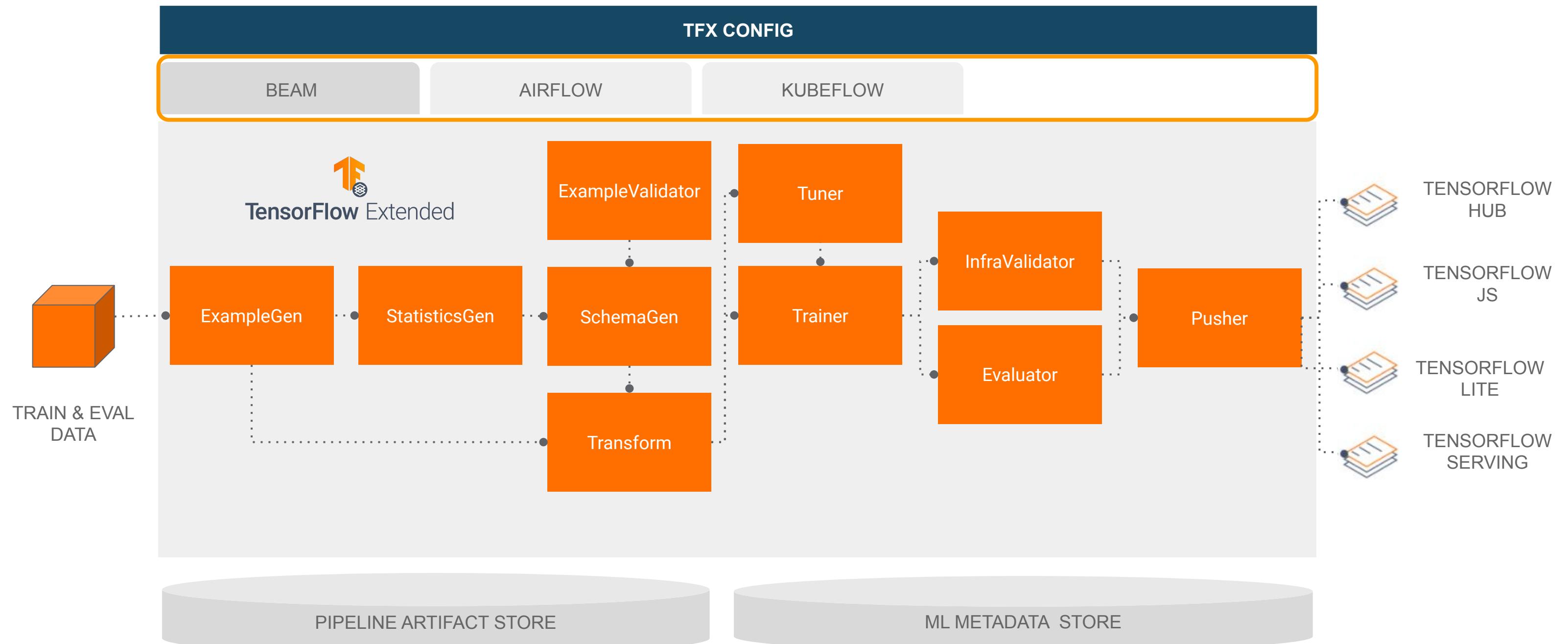
TFX libraries



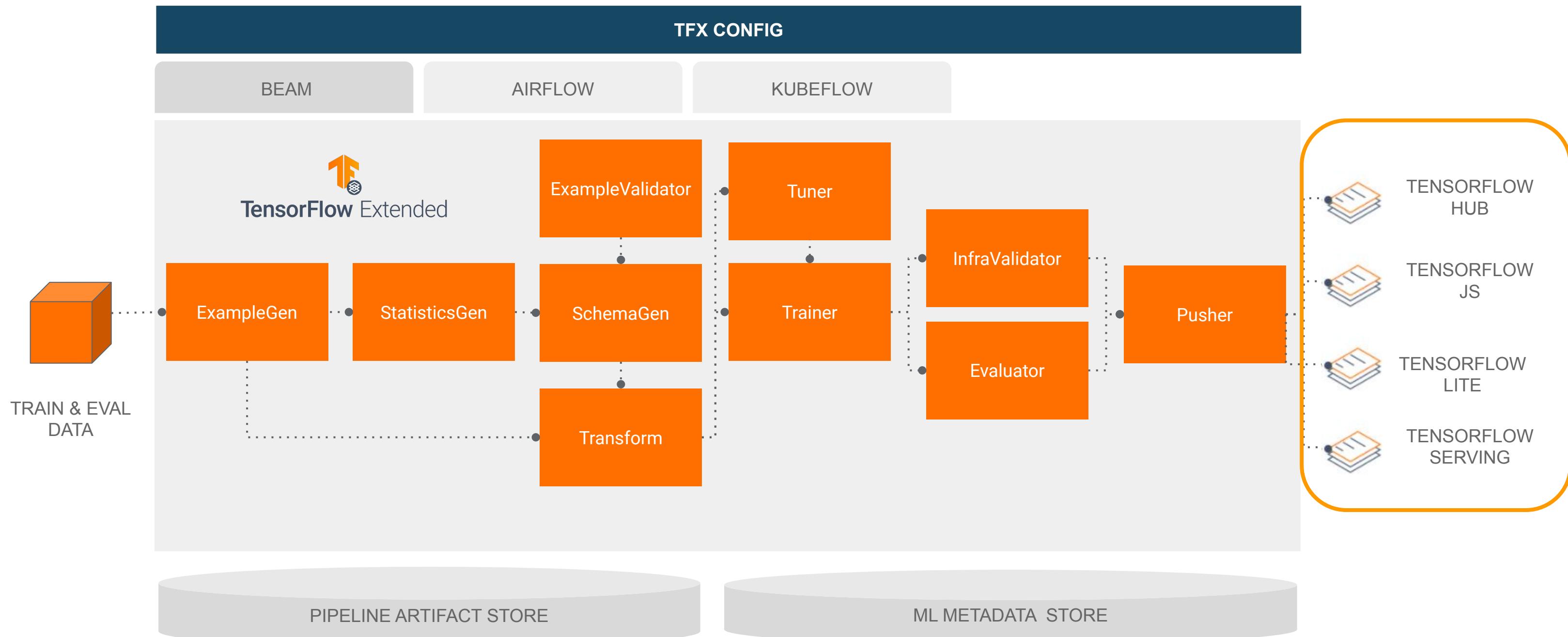
# TFX: Google's production machine learning platform



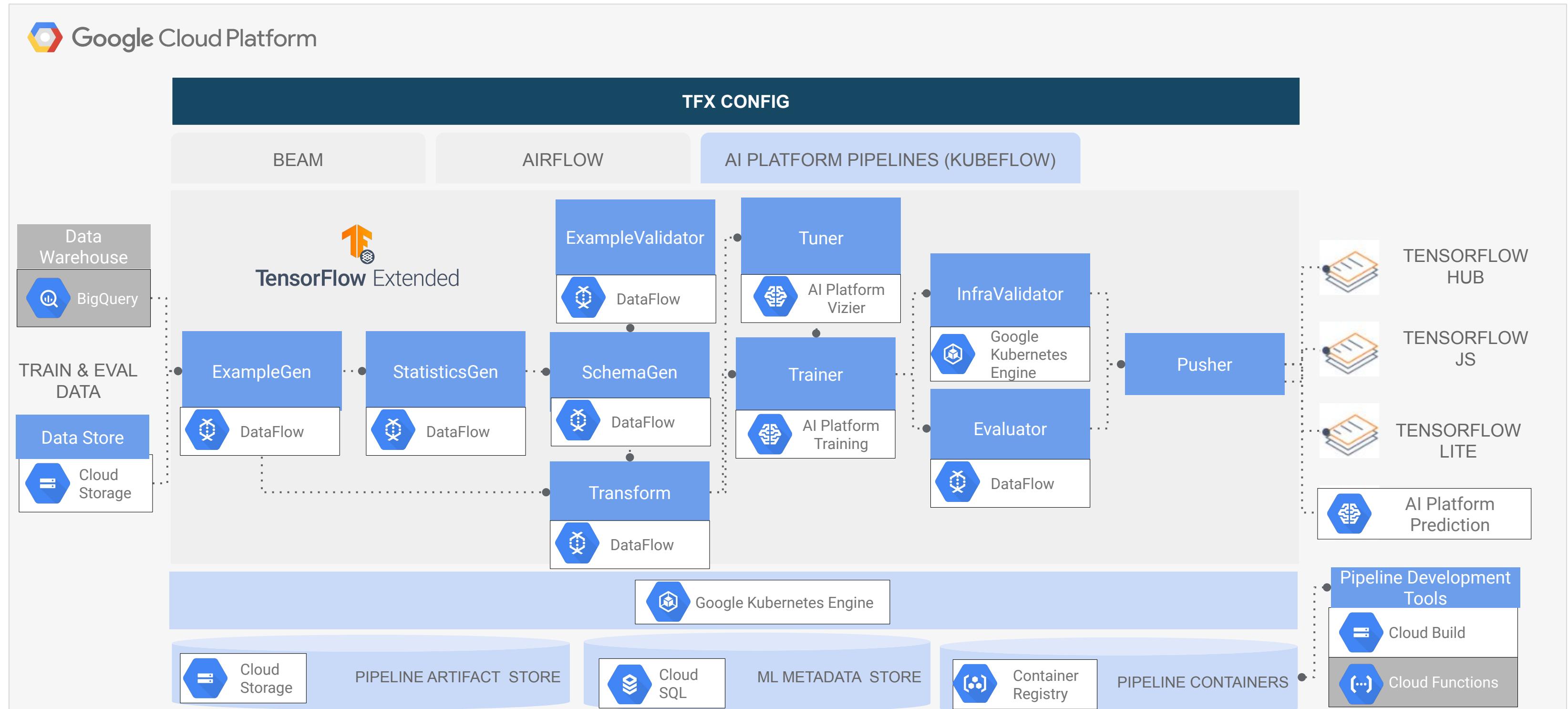
# TFX: Google's production machine learning platform



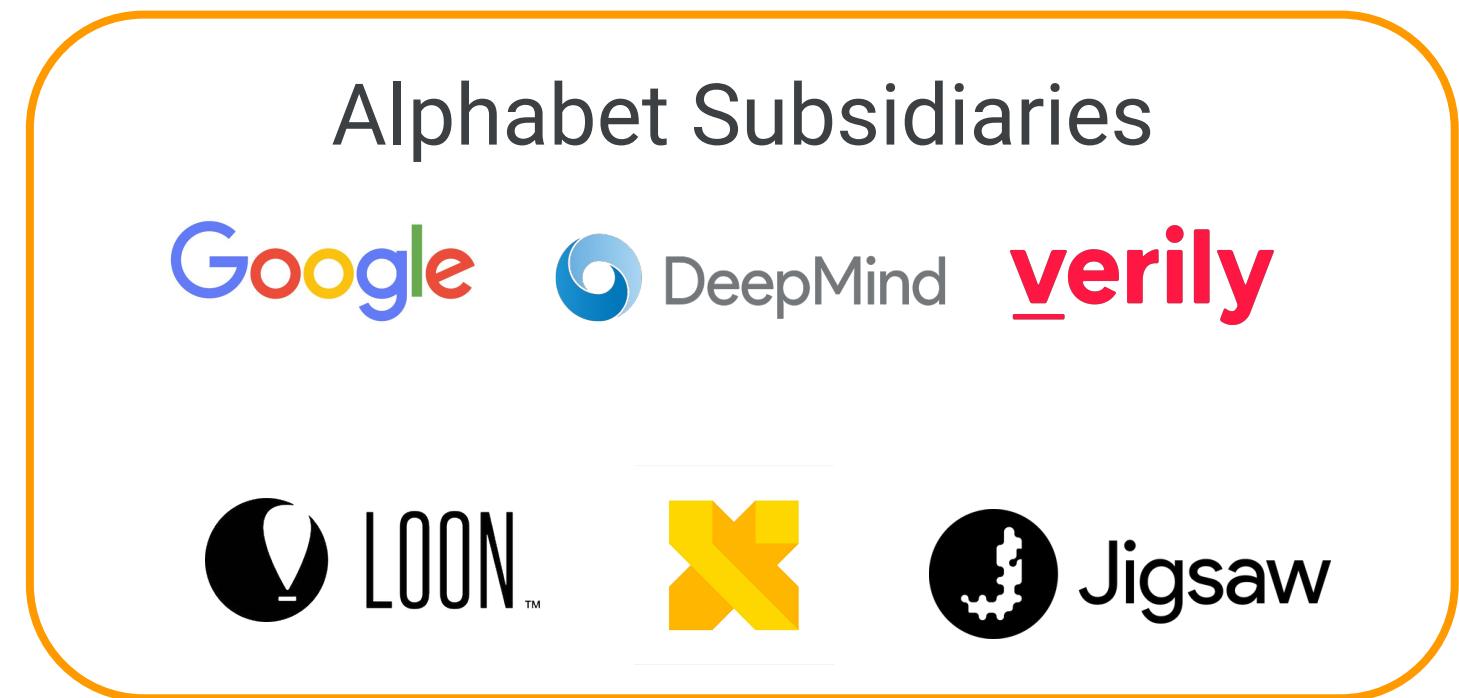
# TFX: Google's production machine learning platform



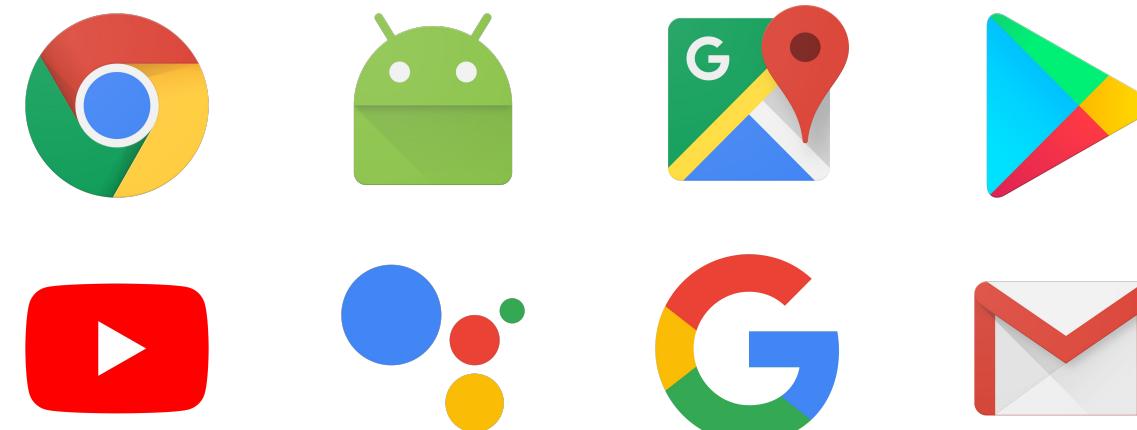
# TFX: Google's production machine learning platform



# TFX powers Alphabet's most important bets and products

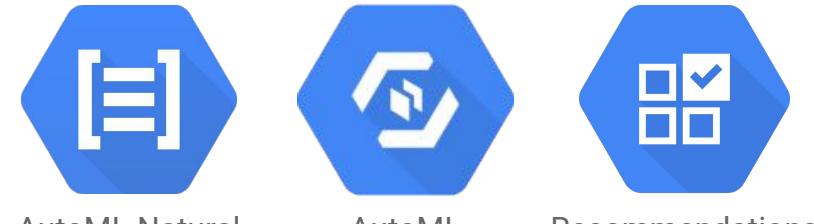


## Google Products



## Google Cloud AI Products

TFX  
powered



TFX  
integrations



# TFX powers Alphabet's most important bets and products

## Alphabet Subsidiaries

 Google  DeepMind  verily

 LOON™



 Jigsaw

## Google Products



## Google Cloud AI Products

TFX  
powered



AutoML  
Natural  
Language



AutoML  
Tables



Recommendations  
AI

TFX  
integrations



AI Platform



Dataflow



BigQuery

# TFX powers Alphabet's most important bets and products

## Alphabet Subsidiaries



## Google Products



## Google Cloud AI Products

TFX  
powered



AutoML  
Natural  
Language



AutoML  
Tables



Recommendations  
AI

TFX  
integrations



AI Platform



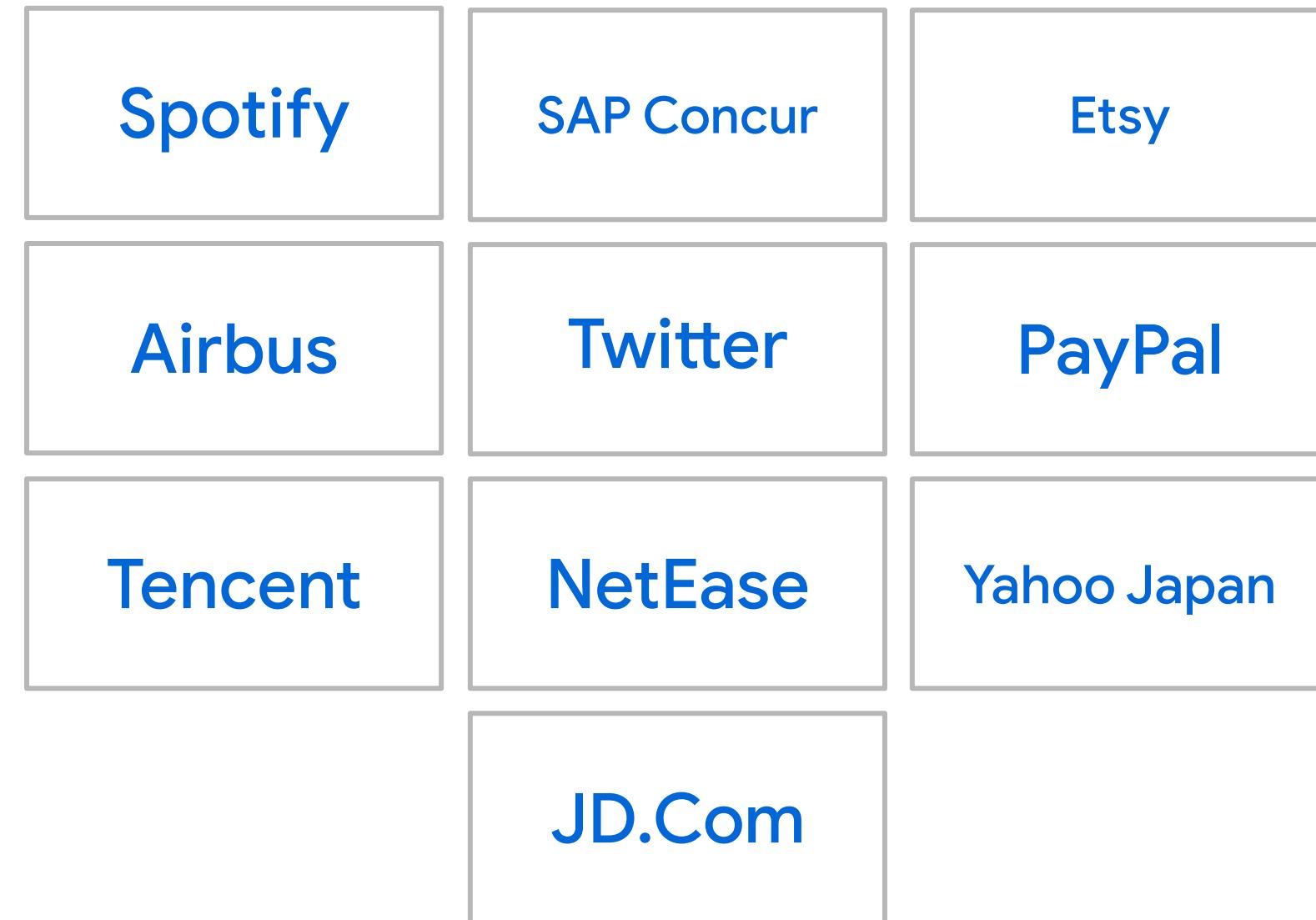
Dataflow



BigQuery

---

# OSS TFX enables global ML use cases

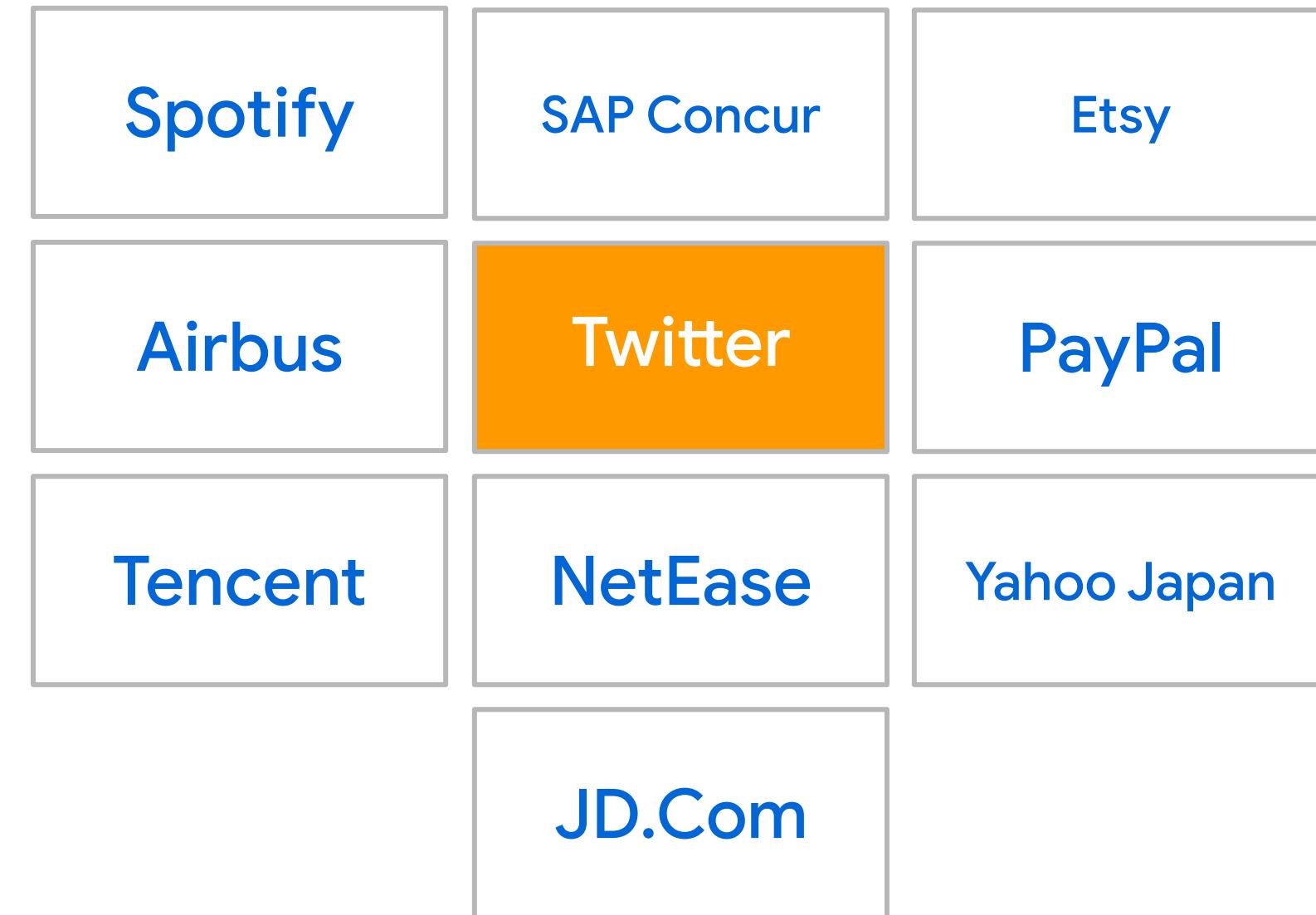


# OSS TFX enables global ML use cases

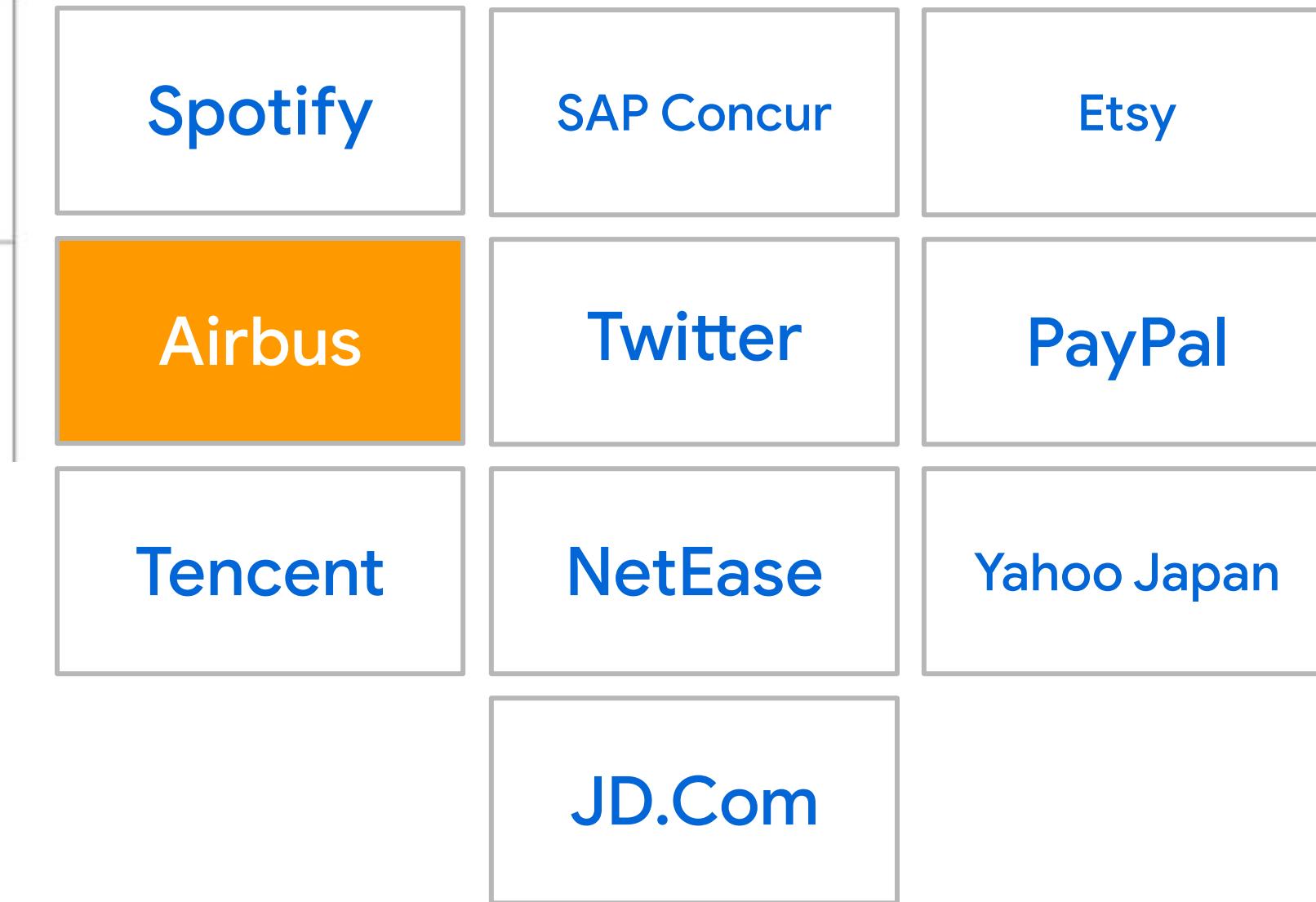
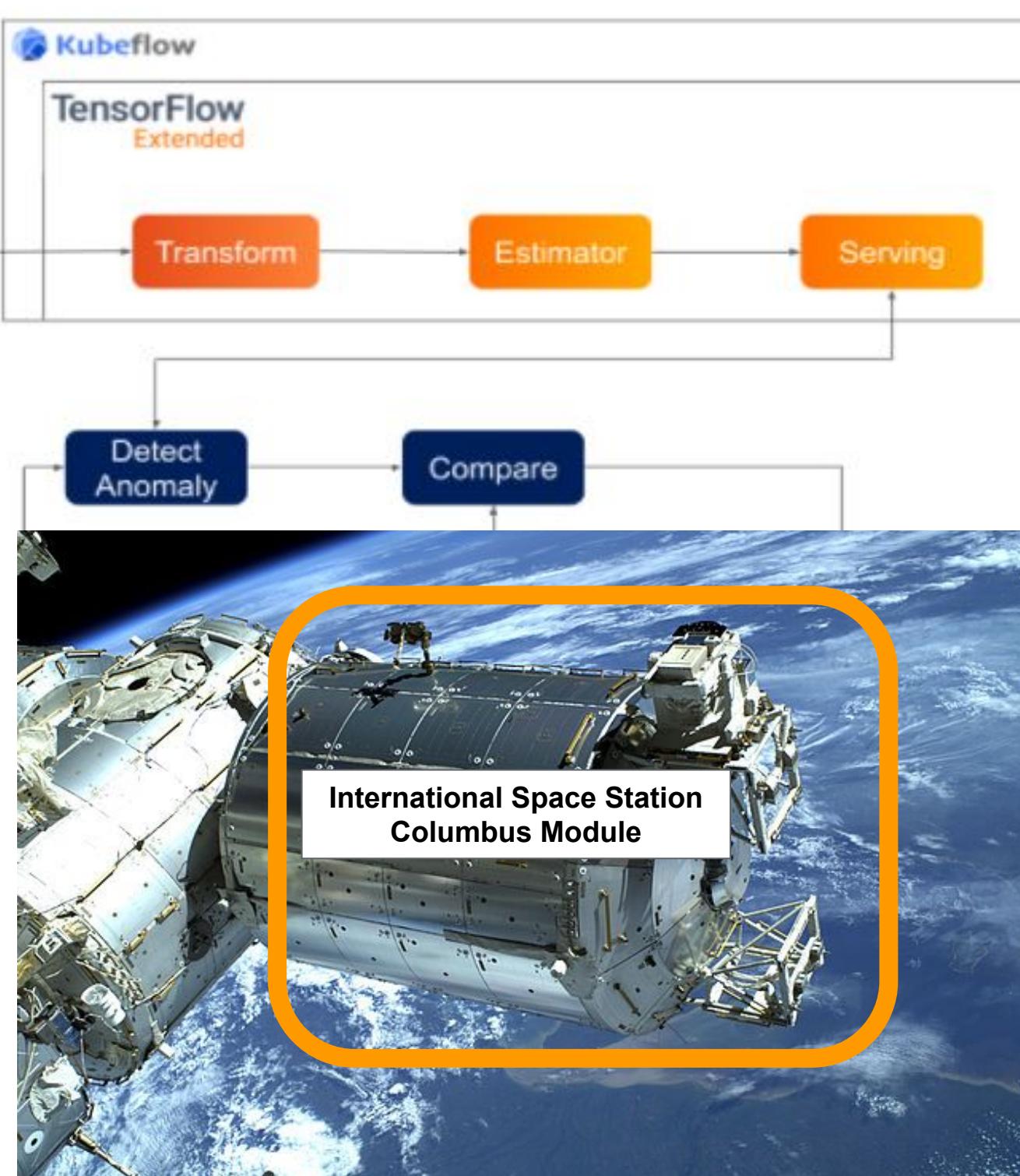
“...we have re-tooled our machine learning platform to use TensorFlow. This yielded significant productivity gains while positioning ourselves to take advantage of the latest industry research.”

**Ranking Tweets with  
TensorFlow - Twitter**

<https://goo.gle/tf-twitter-rank>



# OSS TFX enables global ML use cases



# OSS TFX enables global ML use cases

 BERT Sentiment Analysis



Your sentence appears to be positive.

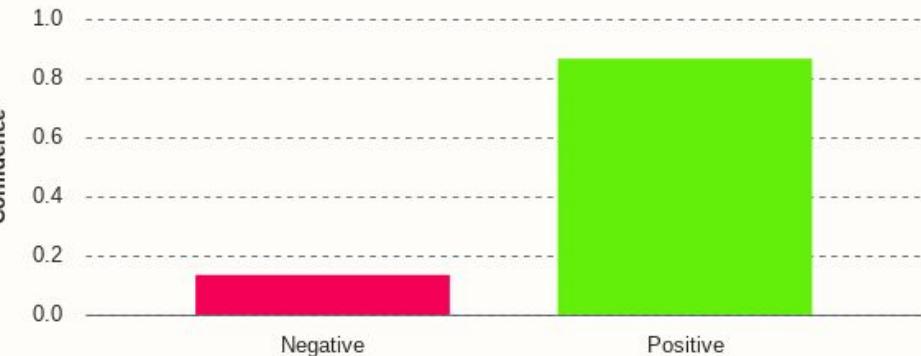
Thanks for the feedback!

Type a sentence to classify its sentiment...

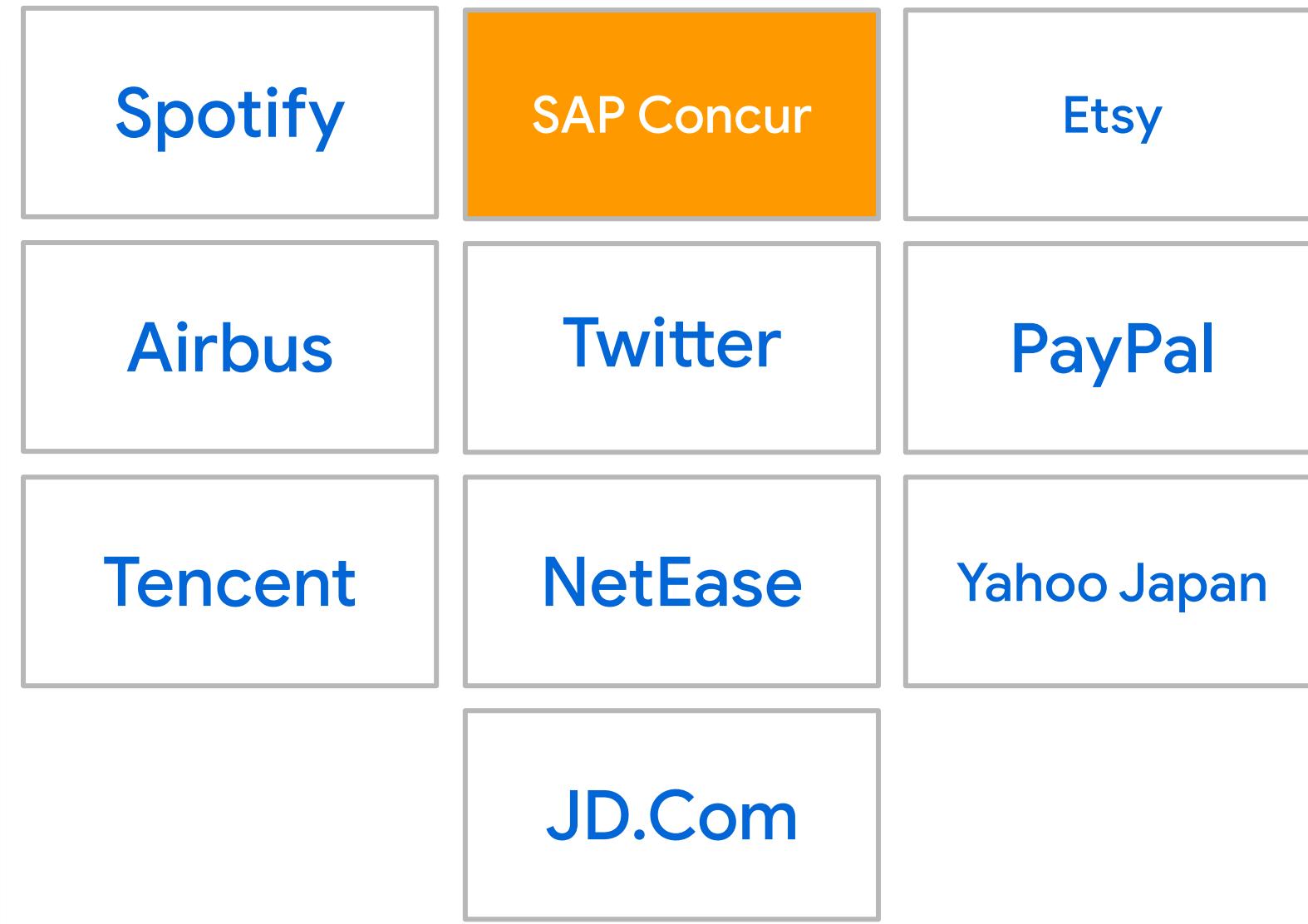
Response time: 456ms

We store the sentence you send to the model to improve the model accuracy.

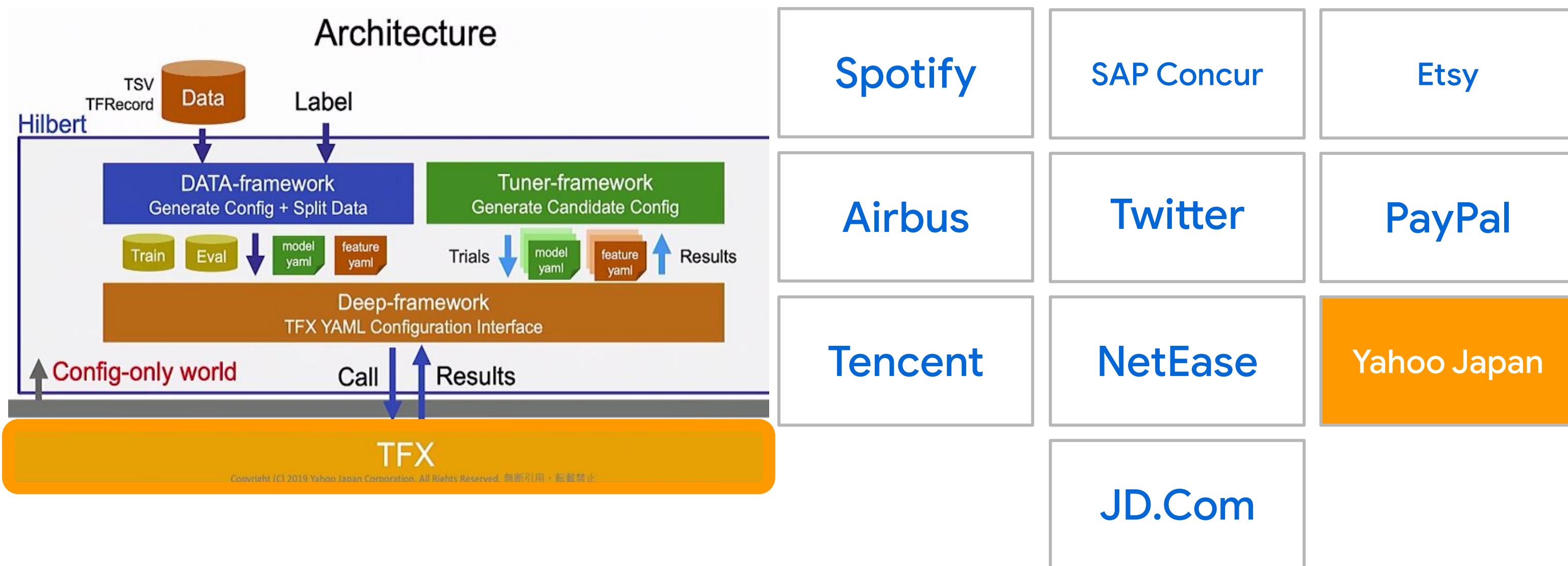
TensorFlow Extended powers SAP Concur BERT deployments



Sentiment	Confidence
Negative	~0.15
Positive	~0.85



# OSS TFX enables global ML use cases



---

# TFX's Lineage

## Sibyl

(2007 - 2019) [goo.gle/sibyl-video](https://goo.gle/sibyl-video)

User-focused, production ML, massive scale,  
(somewhat) flexible

Previously, one of the most widely used E2E ML  
platforms at Google.

## TFX

(2016 - Present) [tensorflow.org/tfx](https://tensorflow.org/tfx)

User-focused, production ML, massive scale,  
**flexible, modular, portable**

The most widely used E2E ML platform at **Alphabet**  
(including Google).

E2E OSS ML platform **on-premise** and on **Google Cloud**.

# TFX has Google's ML best practices built in

Machine Learning: The High Interest Credit Card of Technical Debt. NeurIPS (2015).

TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. KDD (2017).

Data Management Challenges in Production Machine Learning. SIGMOD (2017).

Rules of Machine Learning: Best Practices for ML Engineering. Google AI Web (2017).

Continuous Training for Production ML in the TFX Platform. OpML (2019).

Slice Finder: Automated Data Slicing for Model Validation. ICDE (2019).

Data Validation for Machine Learning. SysML (2019).

The figure displays six academic papers from Google Research, arranged in a grid. Each paper is a rectangular box containing the title, authors, abstract, and a portion of the paper's content.

- KDD 2017 Applied Data Science Paper**  
**TFX: A TensorFlow-Based Production-Scale Machine Learning Platform**  
Denis Baylor, Eric Breck, Heng Li, Salem Haykal, Mustafa Ispir, Clemens Mewald, Akshay Narayan, Steven Euijong Whang, Martin Zinkevich  
D. Sculley, Todd Phillips, {dsculley, tdp} @toddphilli...  
**ABSTRACT**

Creating and maintaining a platform for reliable and deploying machine learning models require orchestration of many components—a learner for models based on training data, modules for analyzing both data as well as models, and finally infrastructure for serving models in production. This becomes challenging when data changes over time and needs to be produced continuously. Unfortunately, orchestration is often done ad hoc using glue code scripts developed by individual teams for specific needs, leading to duplicated effort and fragile system technical debt.

We present TensorFlow Extended (TFX), a general-purpose machine learning platform at Google. By integrating the aforementioned components into one platform, we were able to standardize, simplify the platform configuration, and time to production from the order of months to weeks, undecreased due to providing platform stability that minimizes disruption.

**CCS Concepts**
  - Information systems → Data management
  - Computing methodologies → Machine learning**1. INTRODUCTION**

It is hard to overemphasize the importance of learning in modern computing. More and more companies are adopting machine learning as a tool to glean insights from their data. This is coming for free, remarkably easy to implement, and when applying machine learning specifically where possible. The loops, undecreased due to providing platform stability that minimizes disruption.
- Data Management Challenges in Production Machine Learning**  
Neoklis Polyzotis, Sudip Roy, {npolyzotis, sudip}@google.com  
**ABSTRACT**

This document is intended to help the reader understand the benefit of best practices in machine learning, similar to the Google C++ Style Guide. If you have taken a class on machine learning or have worked with a machine-learned model, then you have probably heard of some of the best practices mentioned here.

**Terminology**
  - Overview
  - Before Machine Learning

Rule #1: Don't be afraid of complexity.  
Rule #2: Make metrics doable.  
Rule #3: Choose machine learning models that fit your needs.

ML: Please Is Your First Discipline
- Continuous Training for Production Machine Learning**  
Denis Baylor, Kevin Haas, Rose Liu, Clemens Mewald, Mitchell Welling, {denisb, khaas, rliu, cmewald, mitchellw}@google.com  
**Abstract**

Large organizations rely increasingly on continuous pipelines in order to keep machine-learned models up-to-date with respect to data. In these disruptions in the pipeline can increase model staleness and thus degrade the quality of downstream services such as these models. In this paper we describe the operational challenges of running continuous pipelines in the Tensorflow Extended (TFX) system that we developed and deployed at Google. We present the main mechanisms in TFX to support this type of production and the lessons learned from the deployment of the platform internally at Google.

**Index Terms**—data slicing, model validation, model architecture, machine learning, continuous training.
- Automated Data Slicing for Model Validation**  
Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Sudip Roy, {yeounohc, timkraska, npolyzotis, sudip}@google.com  
**Abstract**

As machine learning systems become deployed in production, it is important to validate the quality of the data. We focus on the particular problem of slicing data. This is an important problem in model validation because it allows users to analyze the model performance on arbitrary slices, our goal is to find interpretable and large. We propose Slice Finder, which finds slices that are most problematic. Applications include diagnosing model failure and identifying anomalies. Human interaction is crucial. This research is part of a larger trend of research on automated data slicing for machine learning.

**Index Terms**—data slicing, model validation, model architecture, machine learning, continuous training.
- DATA VALIDATION FOR MACHINE LEARNING**  
Eric Breck<sup>1</sup>, Neoklis Polyzotis<sup>1</sup>, Sudip Roy<sup>1</sup>, Steven Euijong Whang<sup>2</sup>, Martin Zinkevich<sup>1</sup>  
**ABSTRACT**

Machine learning is a powerful tool for gleaning knowledge from massive amounts of data. While a great deal of machine learning research has focused on improving the accuracy and efficiency of training and inference algorithms, there is less attention in the equally important problem of monitoring the quality of data fed to machine learning. The importance of this problem is hard to dispute: errors in the input data can nullify any benefits on speed and accuracy for training and inference. This argument points to a data-centric approach to machine learning that treats training and serving data as an important production asset, on par with the algorithm and infrastructure used for learning.

In this paper, we tackle this problem and present a data validation system that is designed to detect anomalies specifically in data fed into machine learning pipelines. This system is deployed in production as an integral part of TFX(Baylor et al., 2017) – an end-to-end machine learning platform at Google. It is used by hundreds of product teams use it to continuously monitor and validate several petabytes of production data per day. We faced several challenges in developing our system, most notably around the ability of ML pipelines to soldier on in the face of unexpected patterns, schema-free data, or training/serving skew. We discuss these challenges, the techniques we used to address them, and the various design choices that we made in implementing the system. Finally, we present evidence from the system’s deployment in production that illustrate the tangible benefits of data validation in the context of ML: early detection of errors, model-quality wins from using better data, savings in engineering hours to debug problems, and a shift towards data-centric workflows in model development.

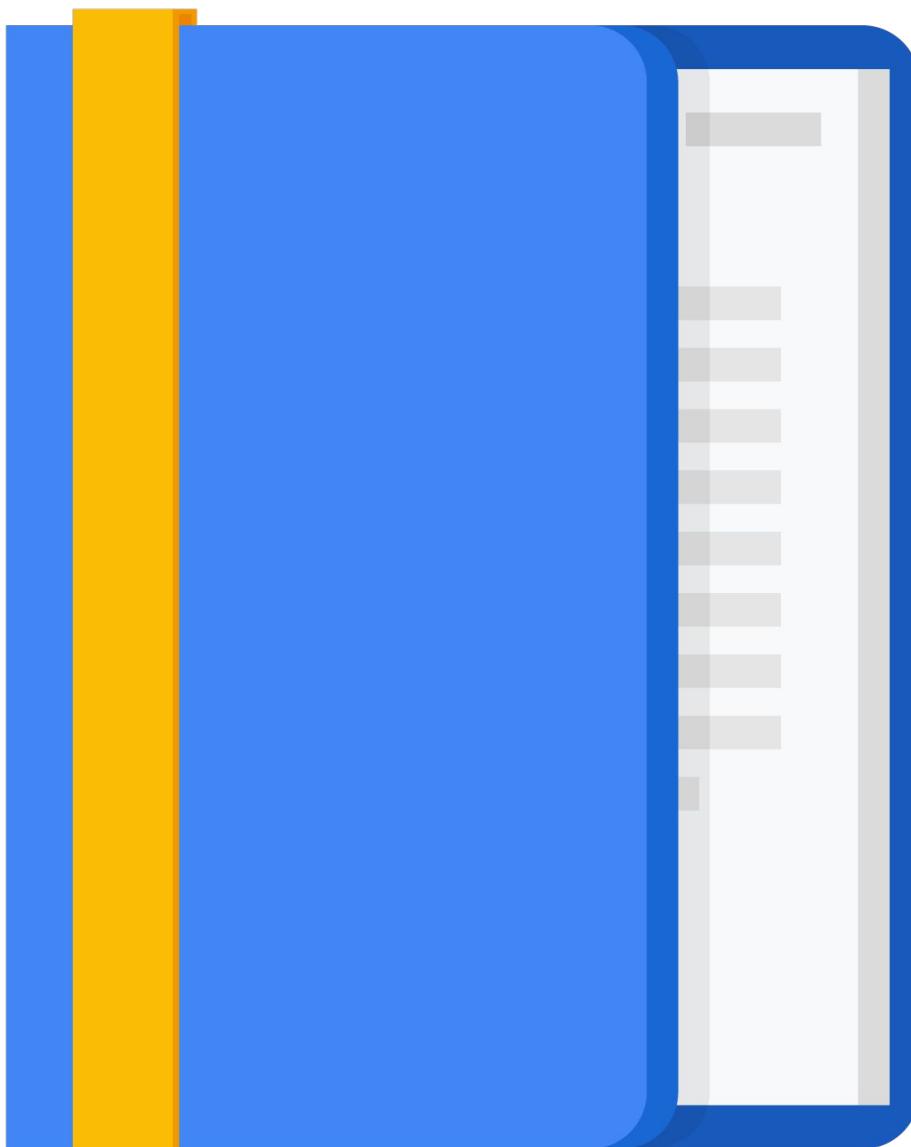
---

# Agenda

TensorFlow Extended (TFX)

TFX standard components

TFX libraries



---

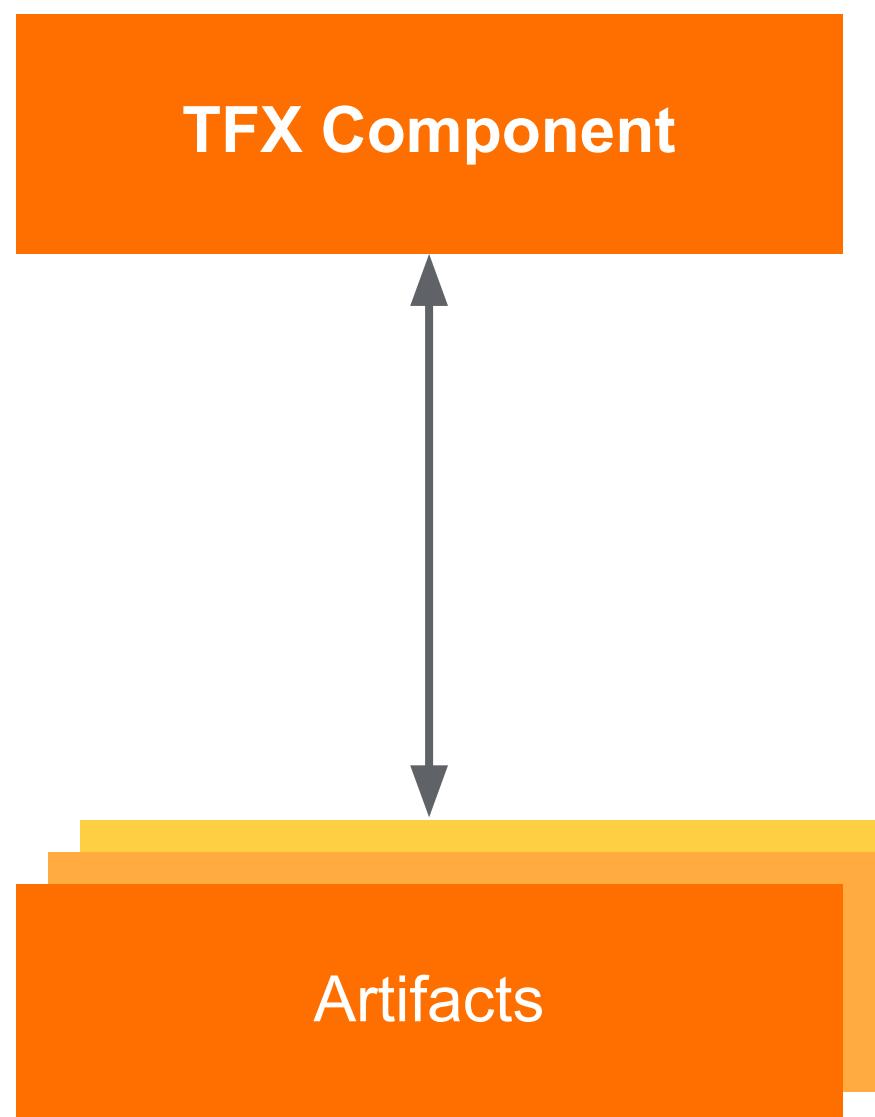
A TFX **component** implements a machine learning task



TFX Component

---

# TFX components produce and consume artifacts



---

# TFX components have 5 elements



**Component specification:** A configuration protocol buffer defines how components communicate with each other via input and output artifact channels and runtime parameters.

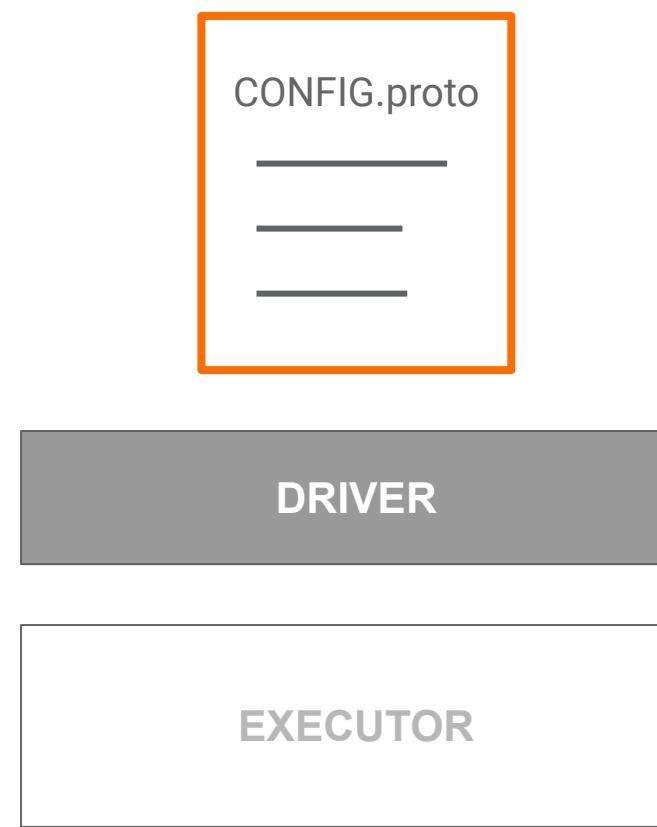
# TFX components have 5 elements



**Component specification:** A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

**Component driver:** A driver coordinates job execution.

# TFX components have 5 elements

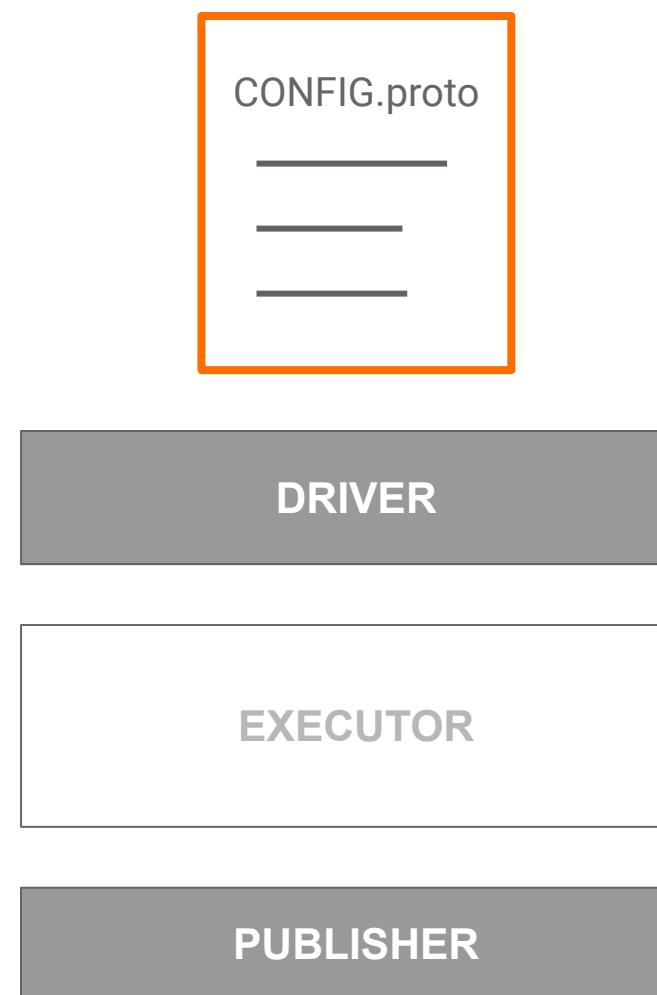


**Component specification:** A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

**Component driver:** A driver coordinates job execution.

**Component executor:** Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

# TFX components produce and consume artifacts



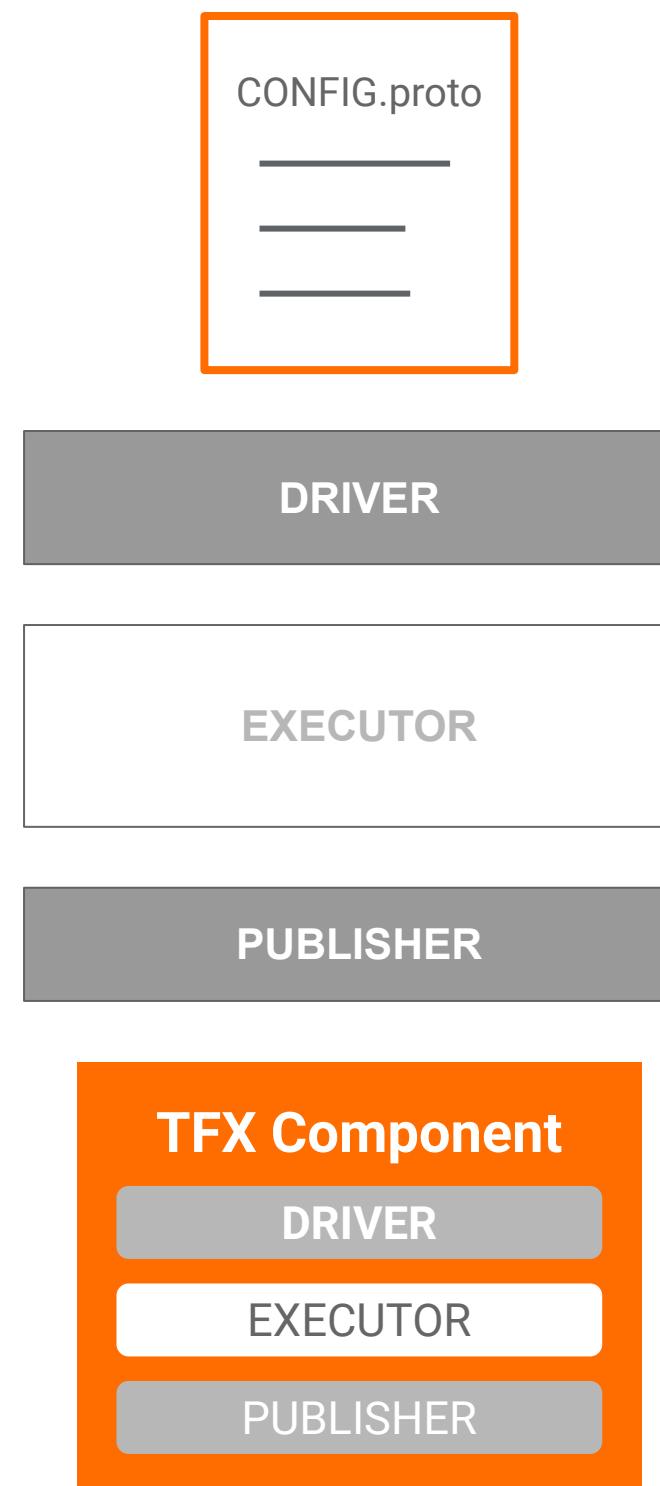
**Component specification:** A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

**Component driver:** A driver coordinates job execution.

**Component executor:** Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

**Component publisher:** Updates ML Metadata store.

# TFX components produce and consume artifacts



**Component specification:** A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

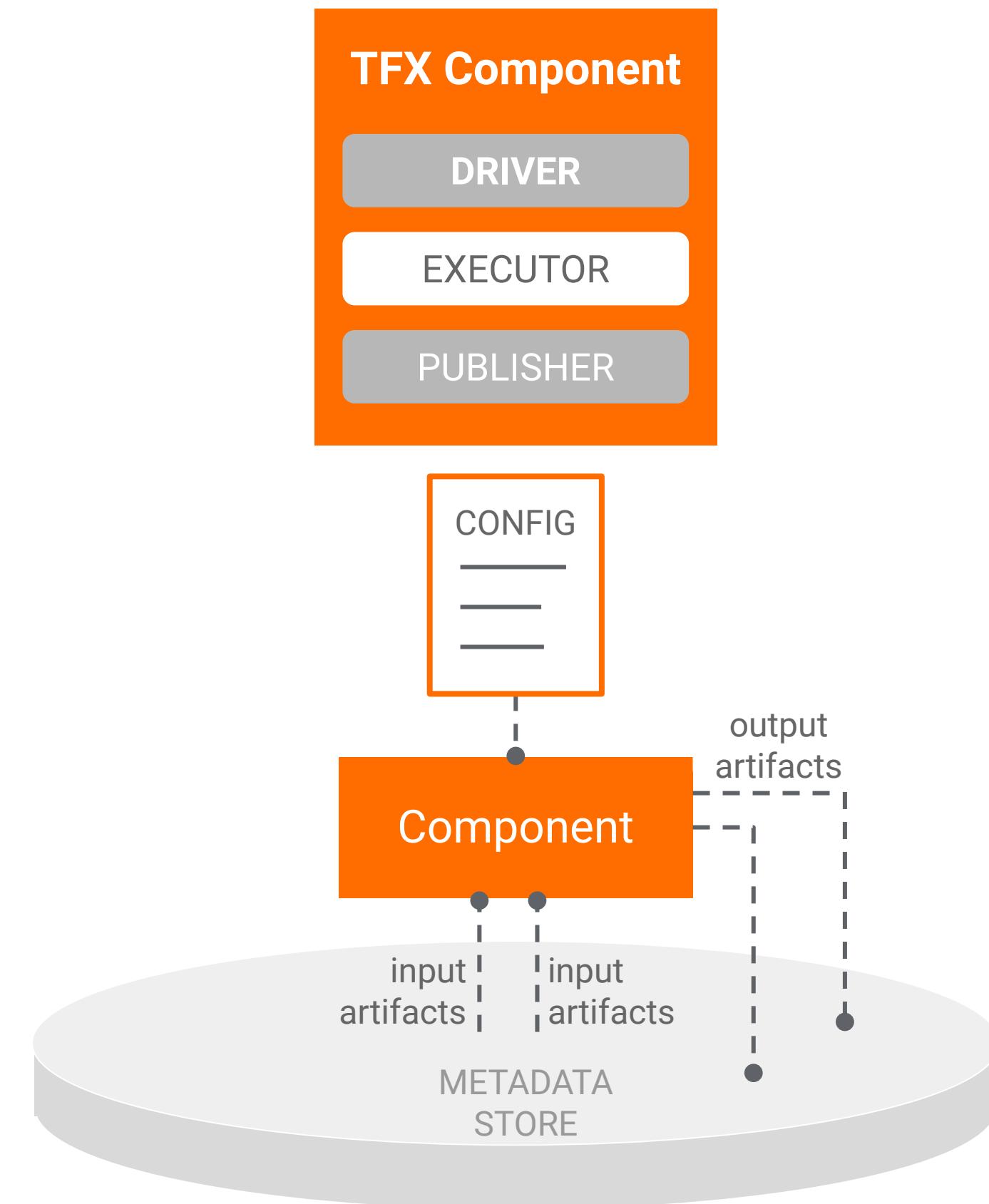
**Component driver:** A driver coordinates job execution.

**Component executor:** Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

**Component publisher:** Updates ML Metadata store.

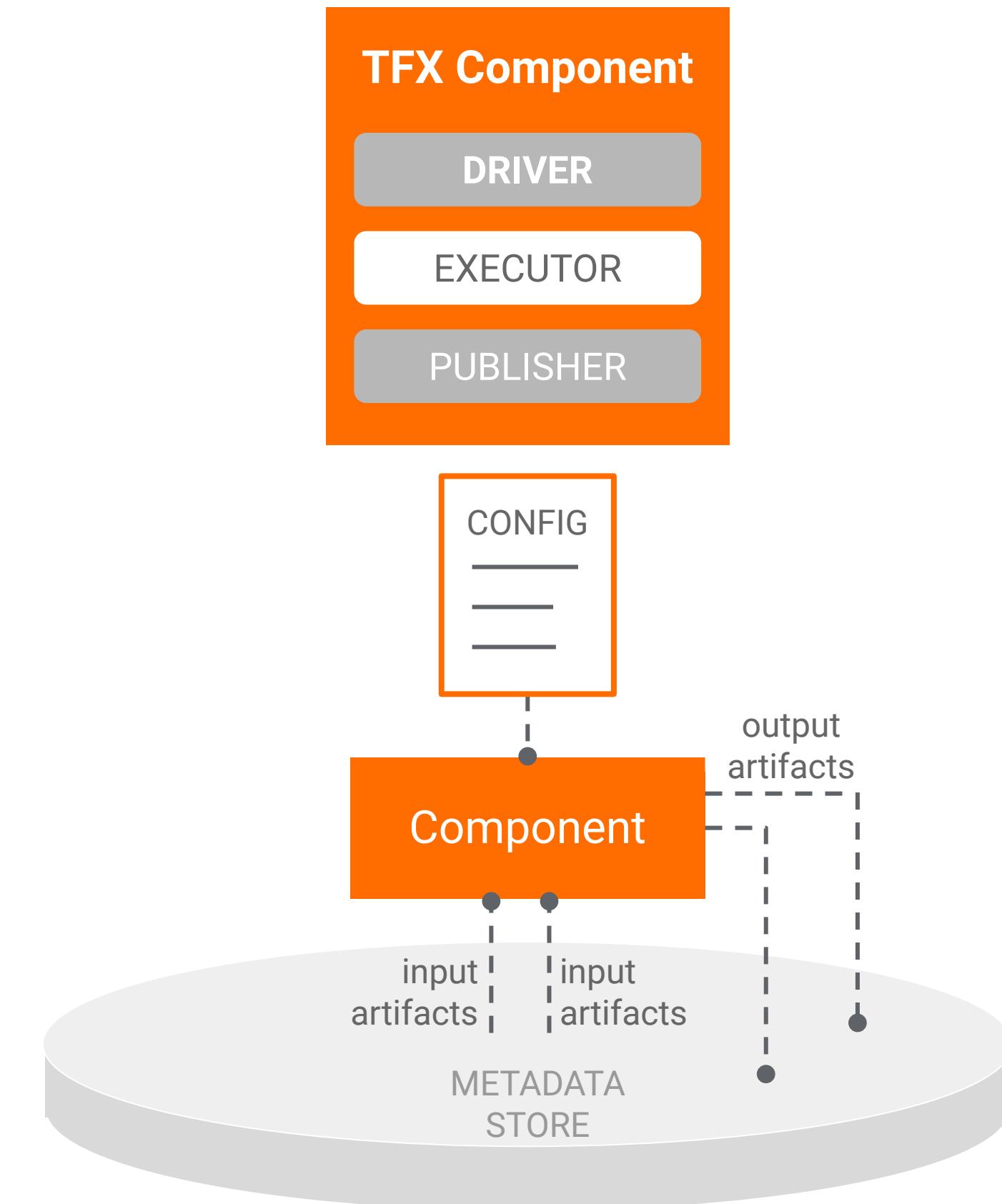
**Component interface:** packages component specification and executor for use in pipeline.

# TFX components at runtime



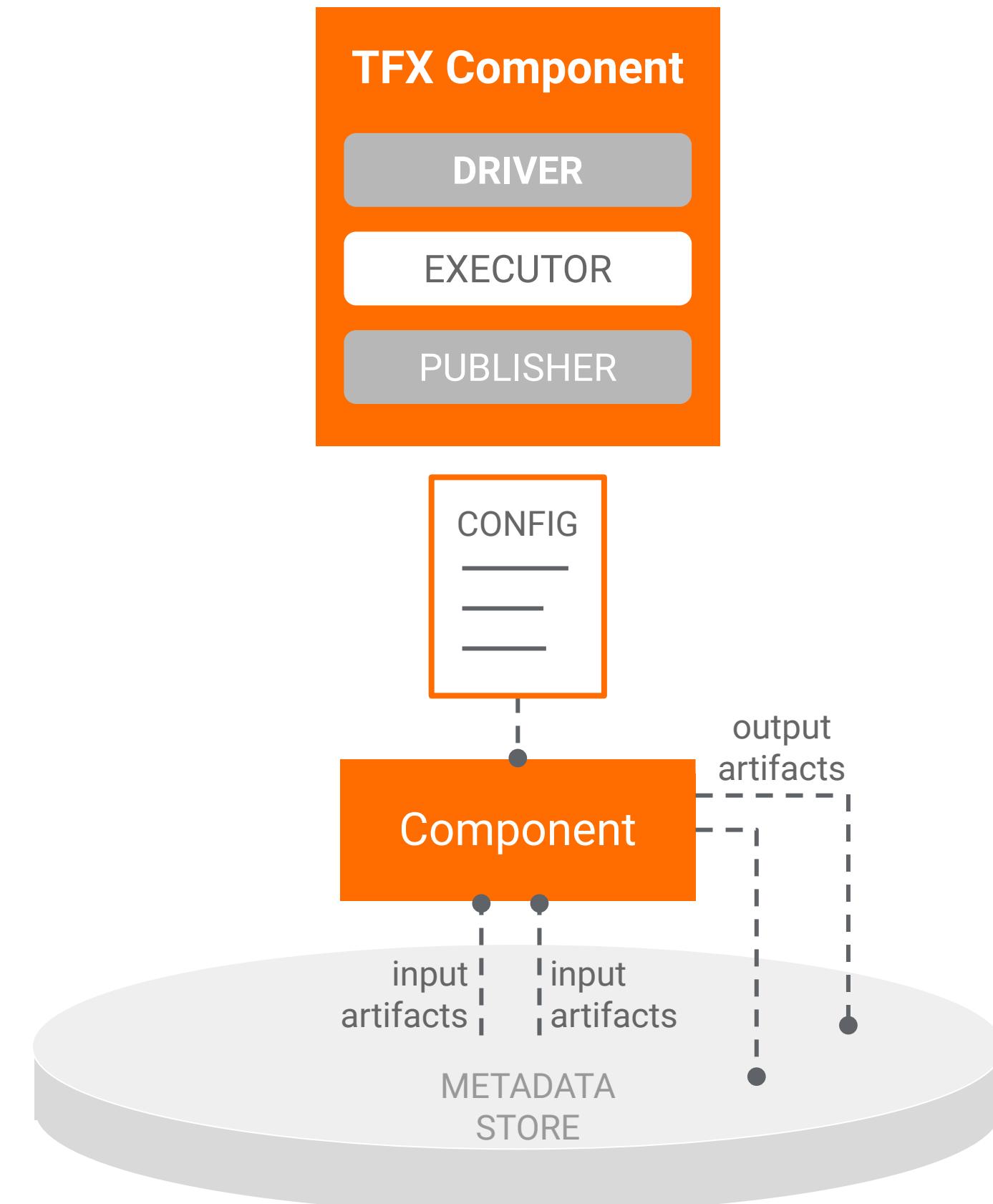
# TFX components at runtime

1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.



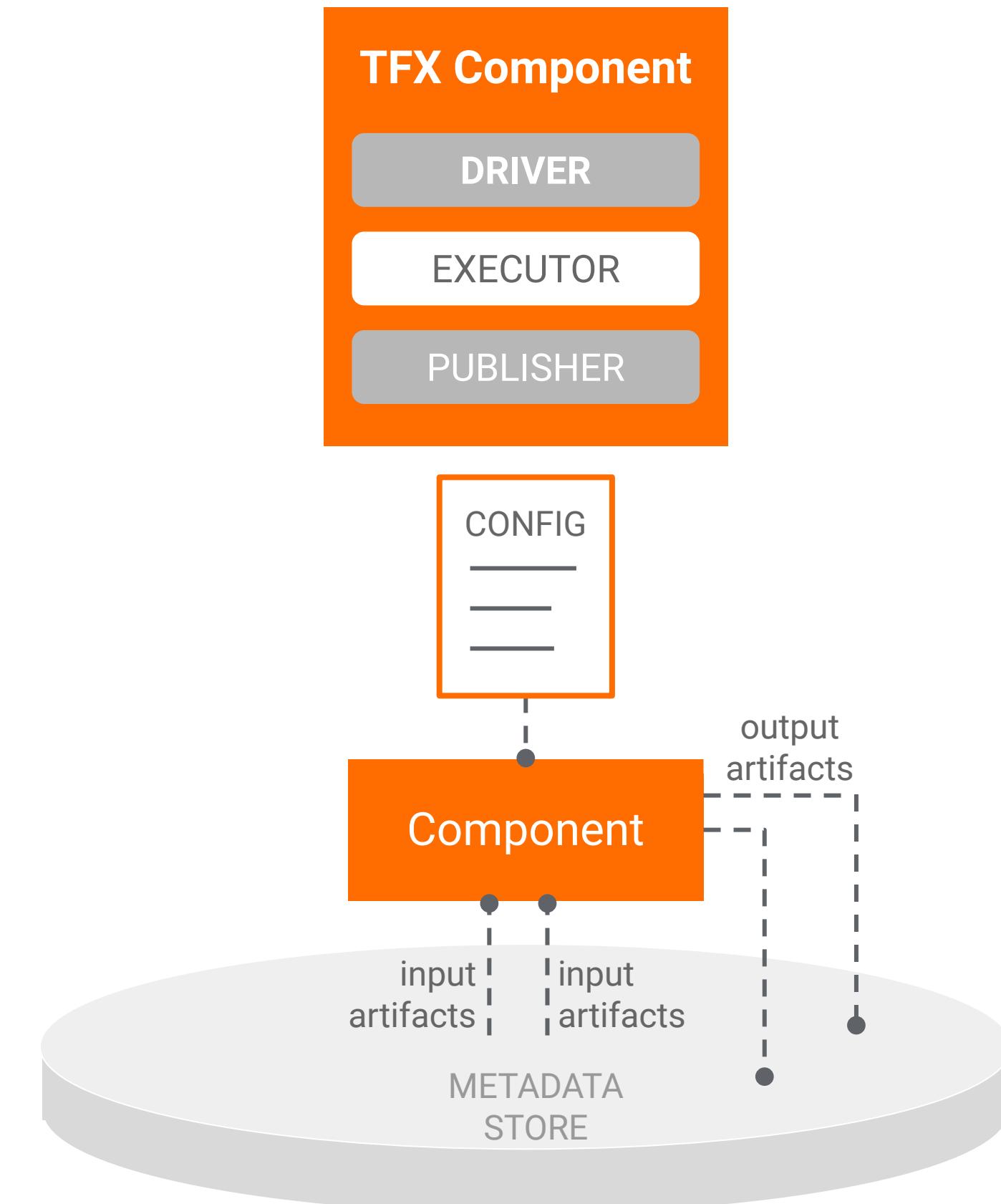
# TFX components at runtime

1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.
2. **Executor** performs computation on artifacts.



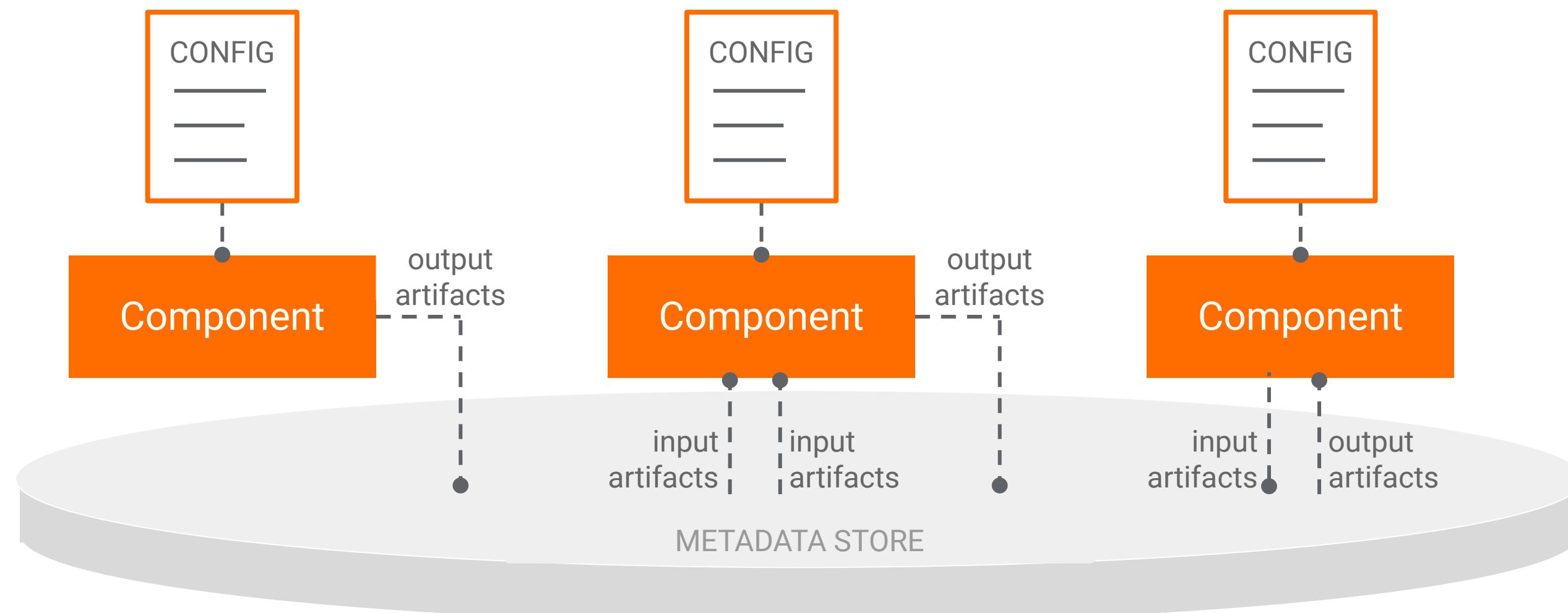
# TFX components at runtime

1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.
2. **Executor** performs computation on artifacts.
3. **Publisher** uses the component specification and executor results to store the component's output artifacts in the metadata store.



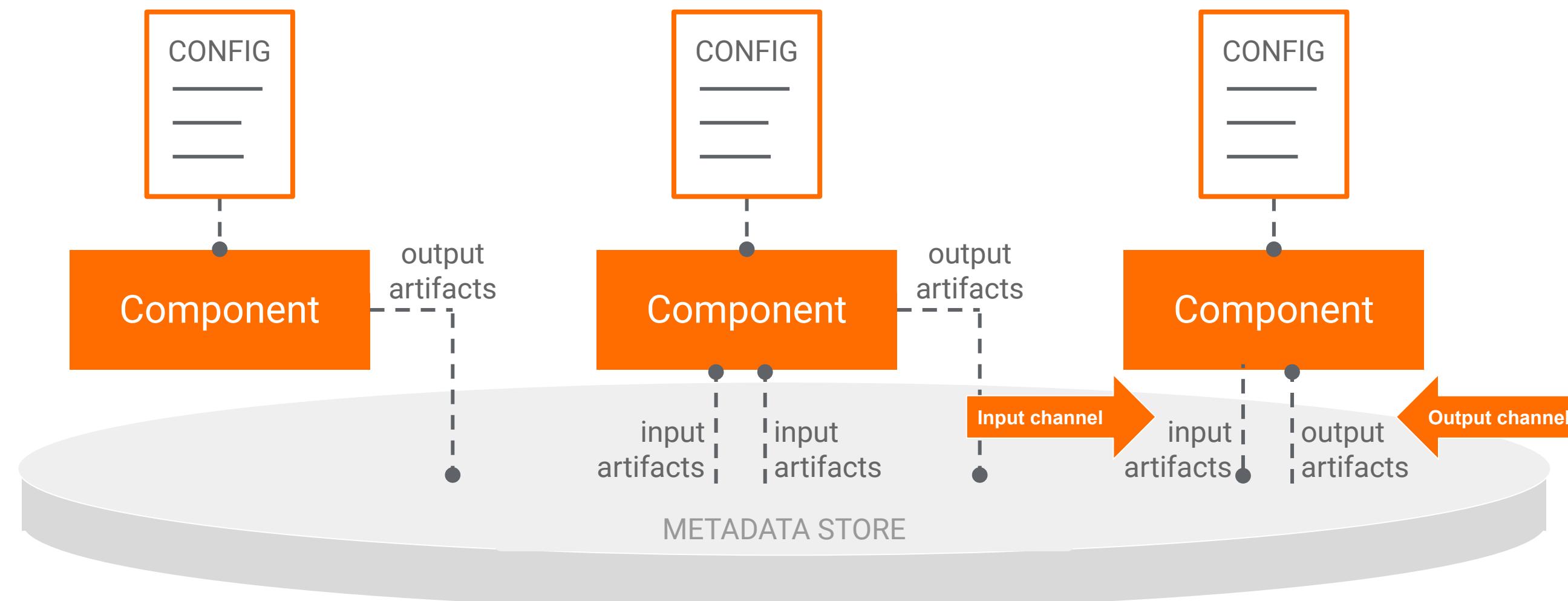
---

A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



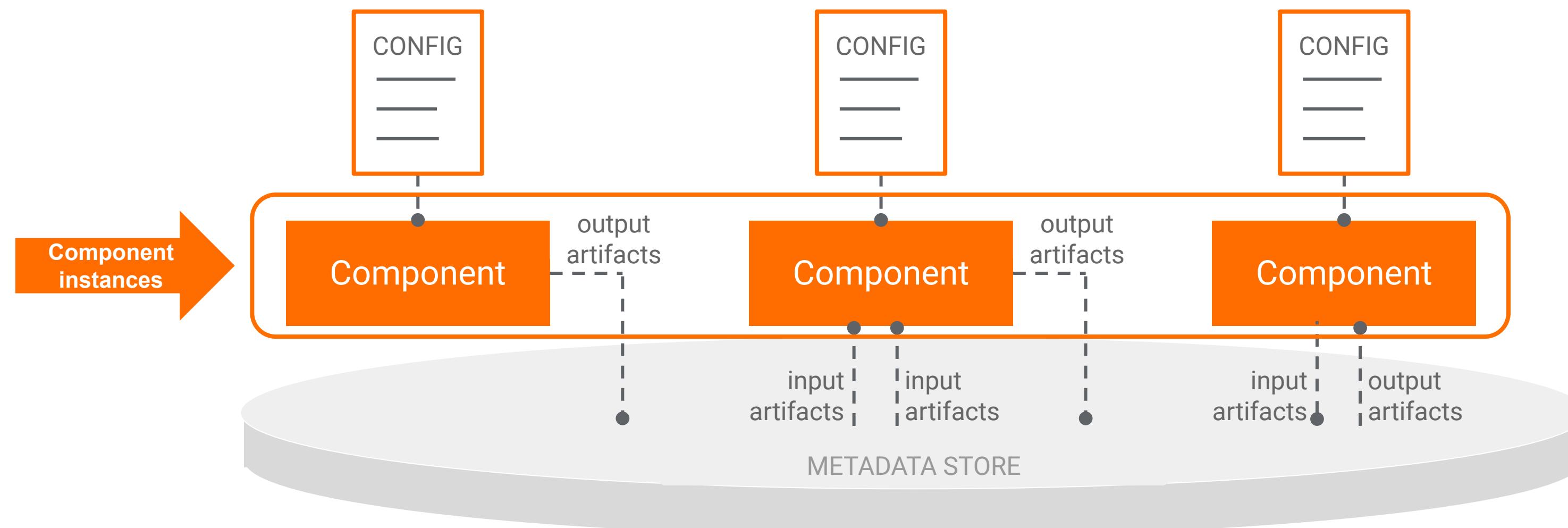
---

A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies

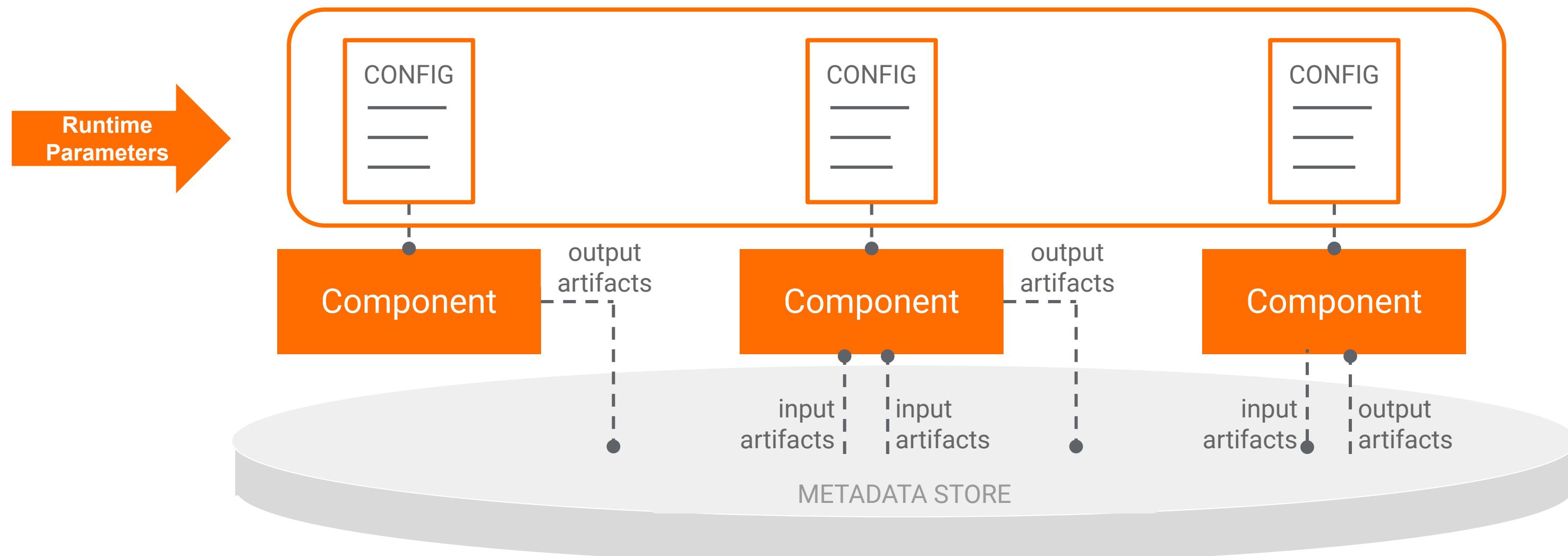


---

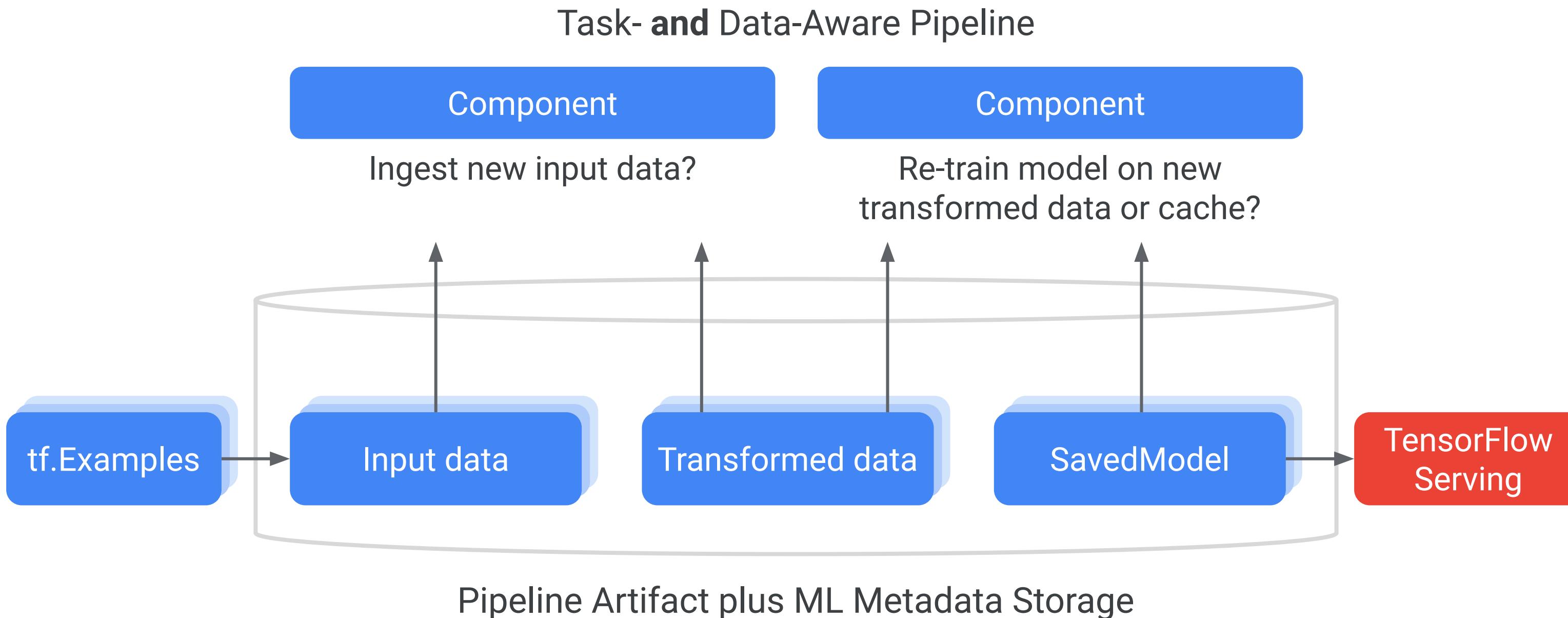
A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies

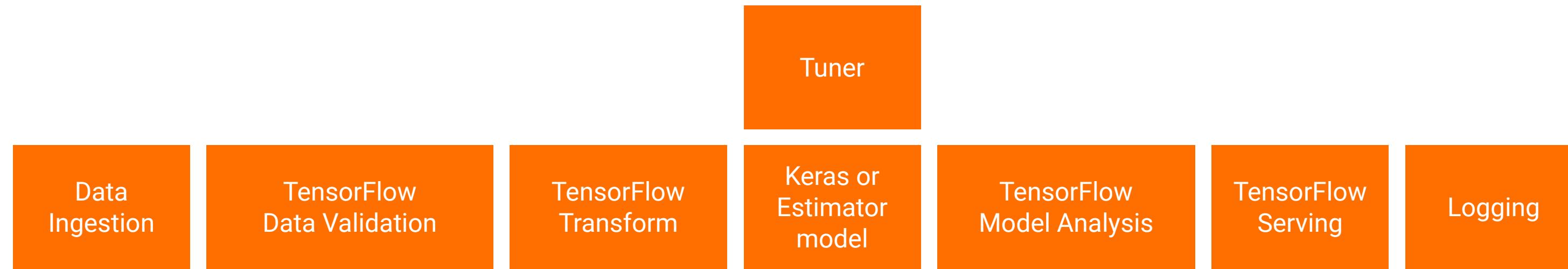


**TFX pipelines use ML Metadata for artifact management and an orchestrator for sequential component execution**



# Horizontal layers coordinate pipeline components

**Integrated Frontend:** pipeline job management, monitoring, debugging, and data/model/evaluation visualization



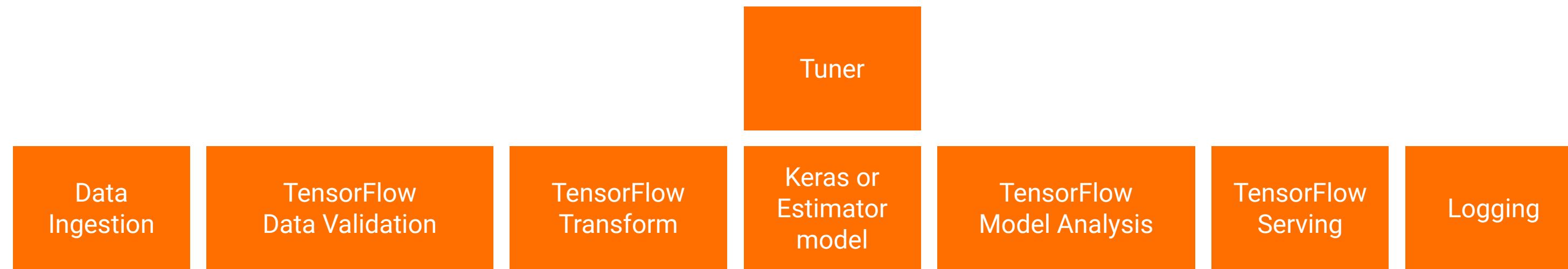
**Orchestrator:** run pipelines using shared code and configurations to distribute component Executor code

**Shared Libraries and Protobufs:** control pipeline garbage collection, data representations, and data access controls

**Pipeline Storage:** artifact storage and ML Metadata for pipeline run and artifact metadata

# Horizontal layers coordinate pipeline components

**Integrated Frontend:** pipeline job management, monitoring, debugging, and data/model/evaluation visualization



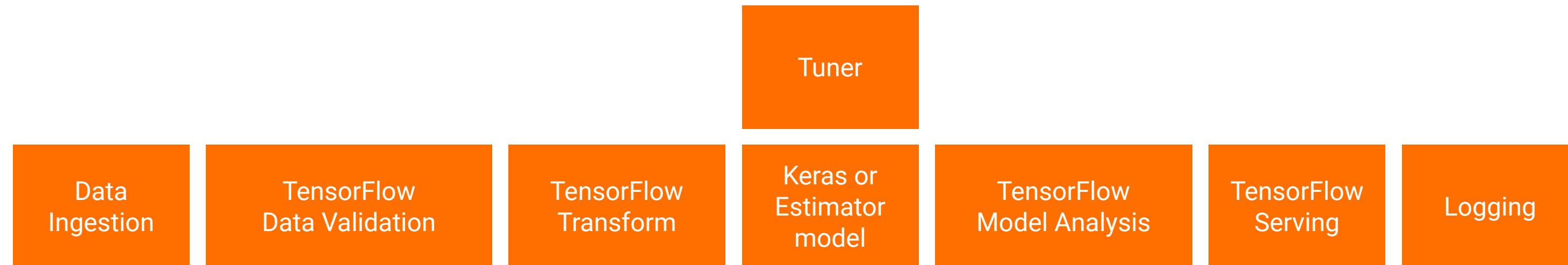
**Orchestrator:** run pipelines using shared code and configurations to distribute component Executor code

**Shared Libraries and Protobufs:** control pipeline garbage collection, data representations, and data access controls

**Pipeline Storage:** artifact storage and ML Metadata for pipeline run and artifact metadata

# Horizontal layers coordinate pipeline components

**Integrated Frontend:** pipeline job management, monitoring, debugging, and data/model/evaluation visualization



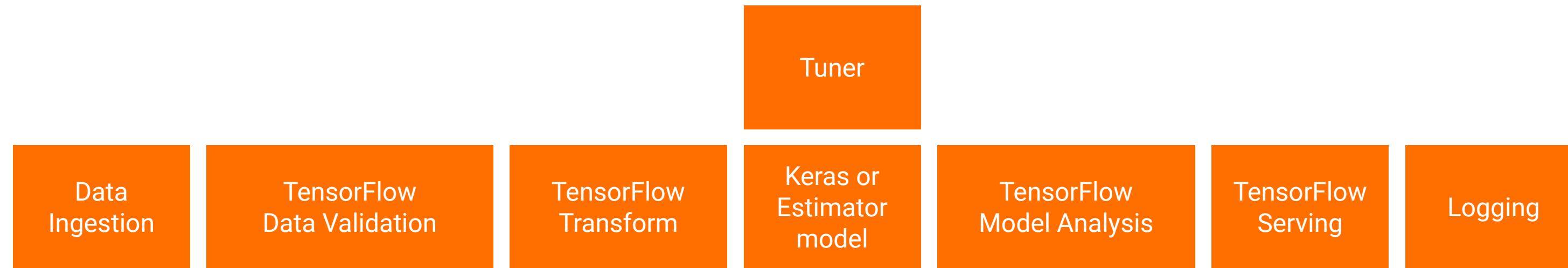
**Orchestrator:** run pipelines using shared code and configurations to distribute component Executor code

**Shared Libraries and Protobufs:** control pipeline garbage collection, data representations, and data access controls

**Pipeline Storage:** artifact storage and ML Metadata for pipeline run and artifact metadata

# Horizontal layers coordinate pipeline components

**Integrated Frontend:** pipeline job management, monitoring, debugging, and data/model/evaluation visualization



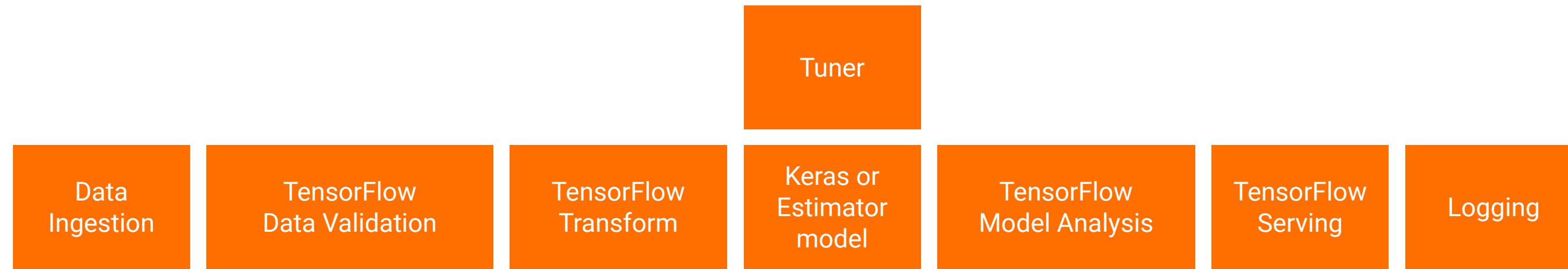
**Orchestrator:** run pipelines using shared code and configurations to distribute component Executor code

**Shared Libraries and Protobufs:** control pipeline garbage collection, data representations, and data access controls

**Pipeline Storage:** artifact storage and ML Metadata for pipeline run and artifact metadata

# Horizontal layers coordinate pipeline components

**Integrated Frontend:** pipeline job management, monitoring, debugging, and data/model/evaluation visualization



**Orchestrator:** run pipelines using shared code and configurations to distribute component Executor code

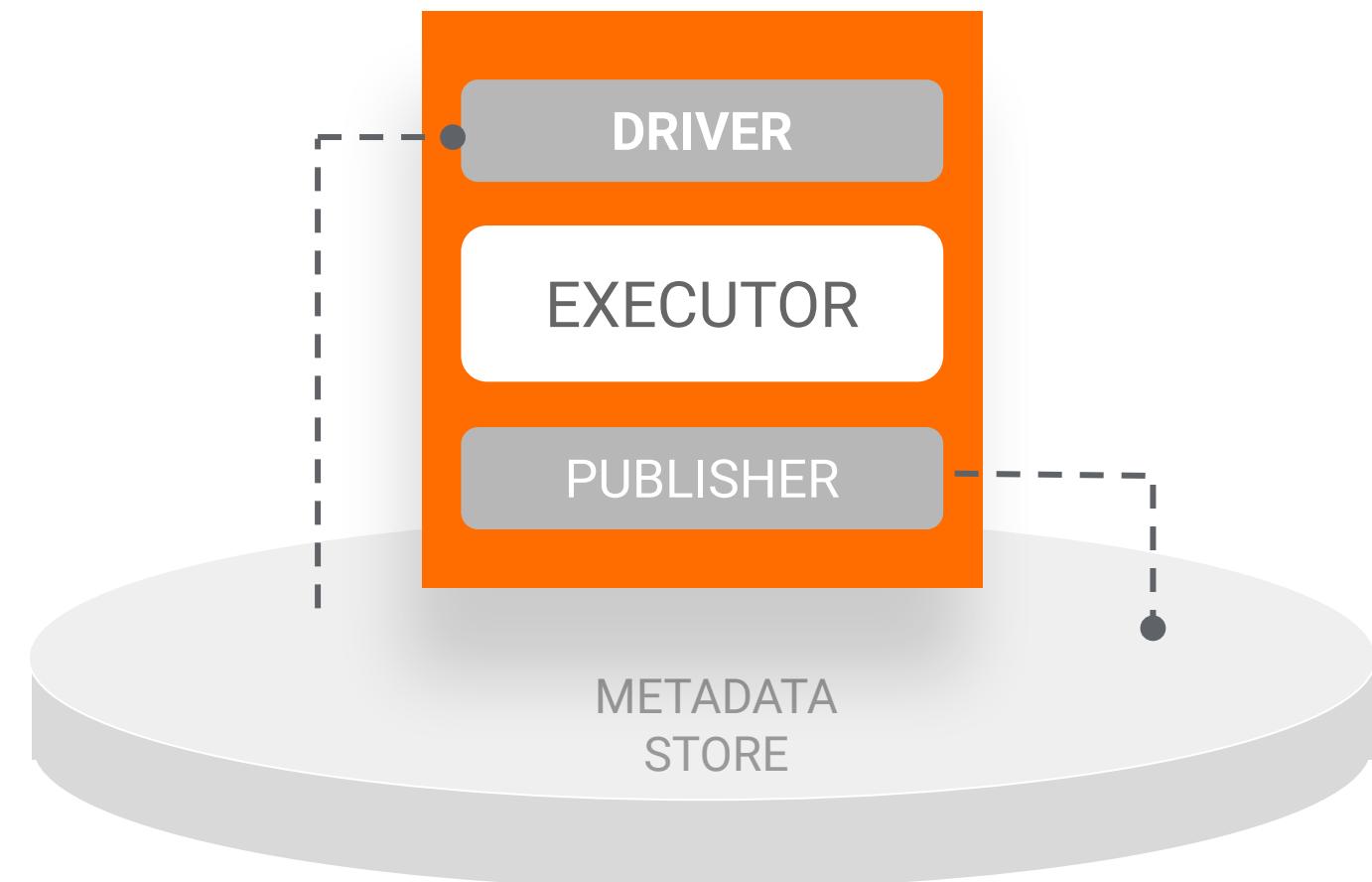
**Shared Libraries and Protobufs:** control pipeline garbage collection, data representations, and data access controls

**Pipeline Storage:** artifact storage and ML Metadata for pipeline run and artifact metadata

---

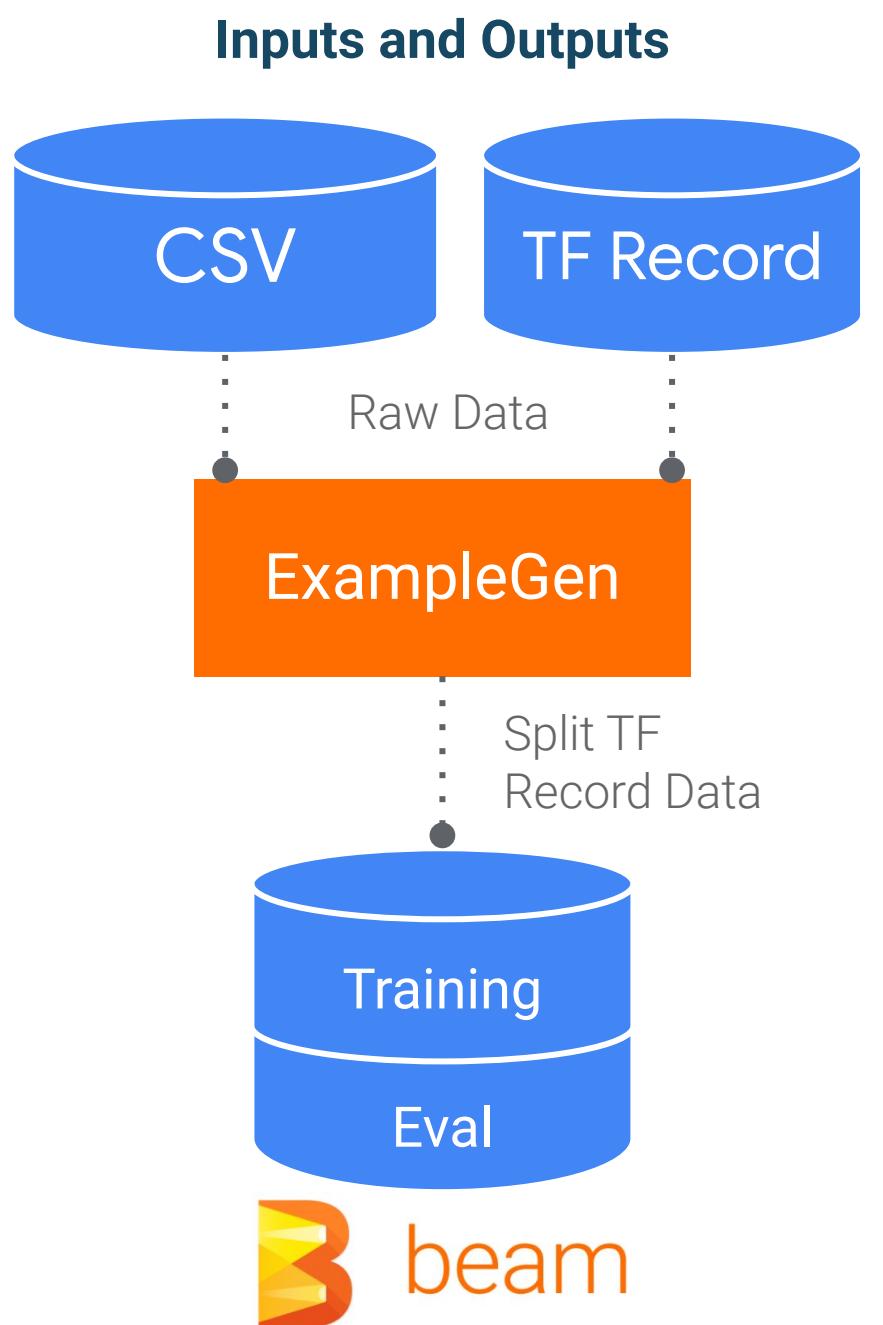
# TFX standard components

Compose ML pipelines faster with reusable components pre-configured with production best practices.

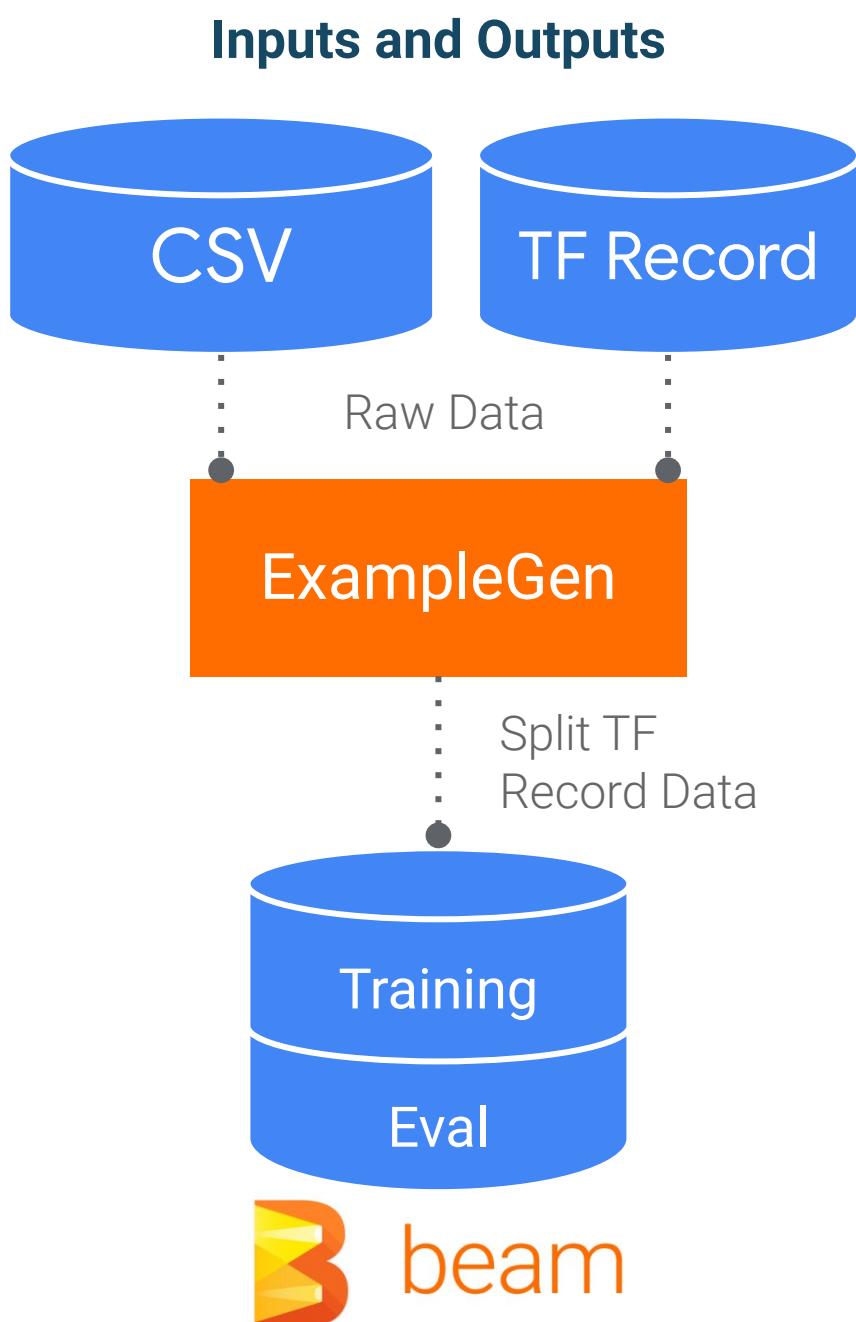


---

# Component: ExampleGen



# Component: ExampleGen

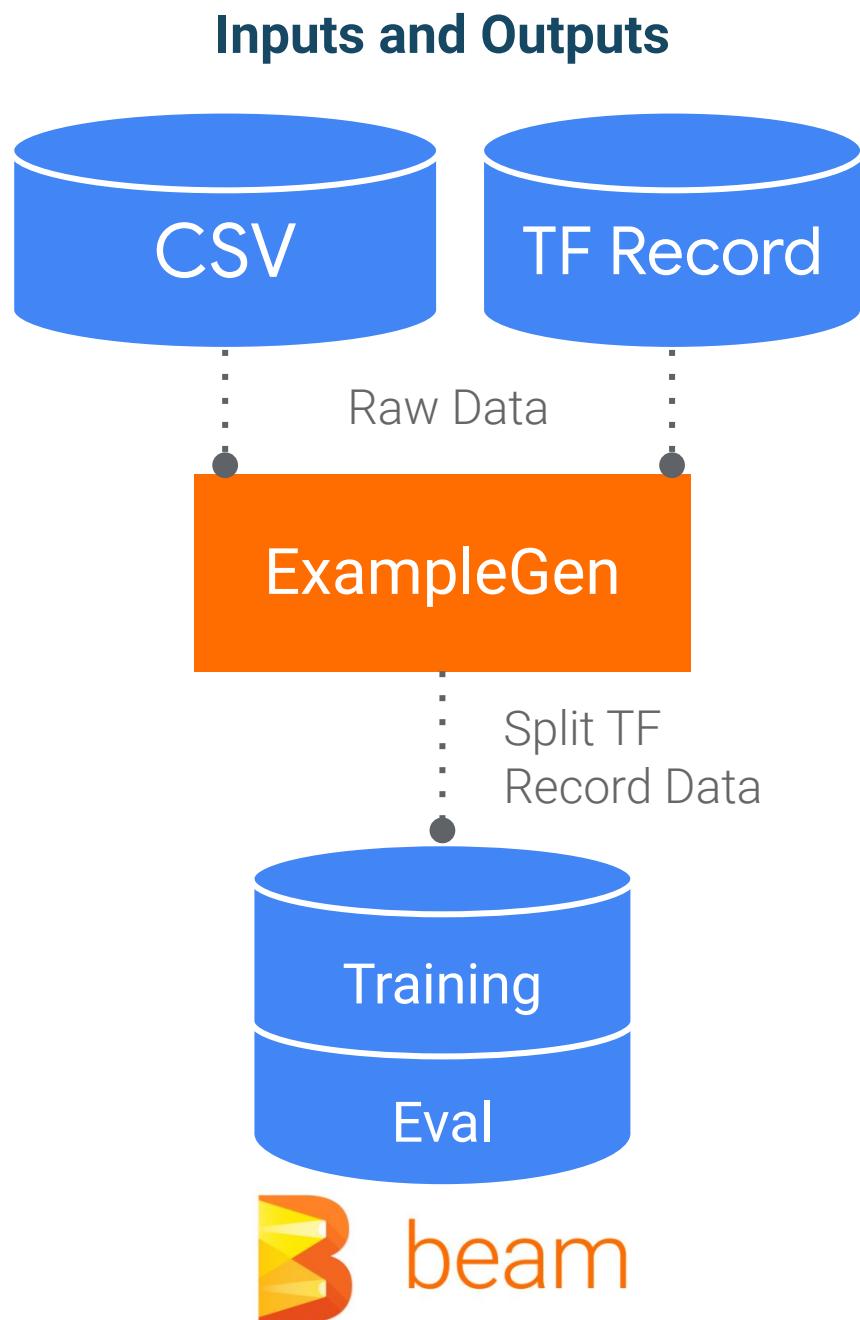


## Configuration

```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

# Component: ExampleGen



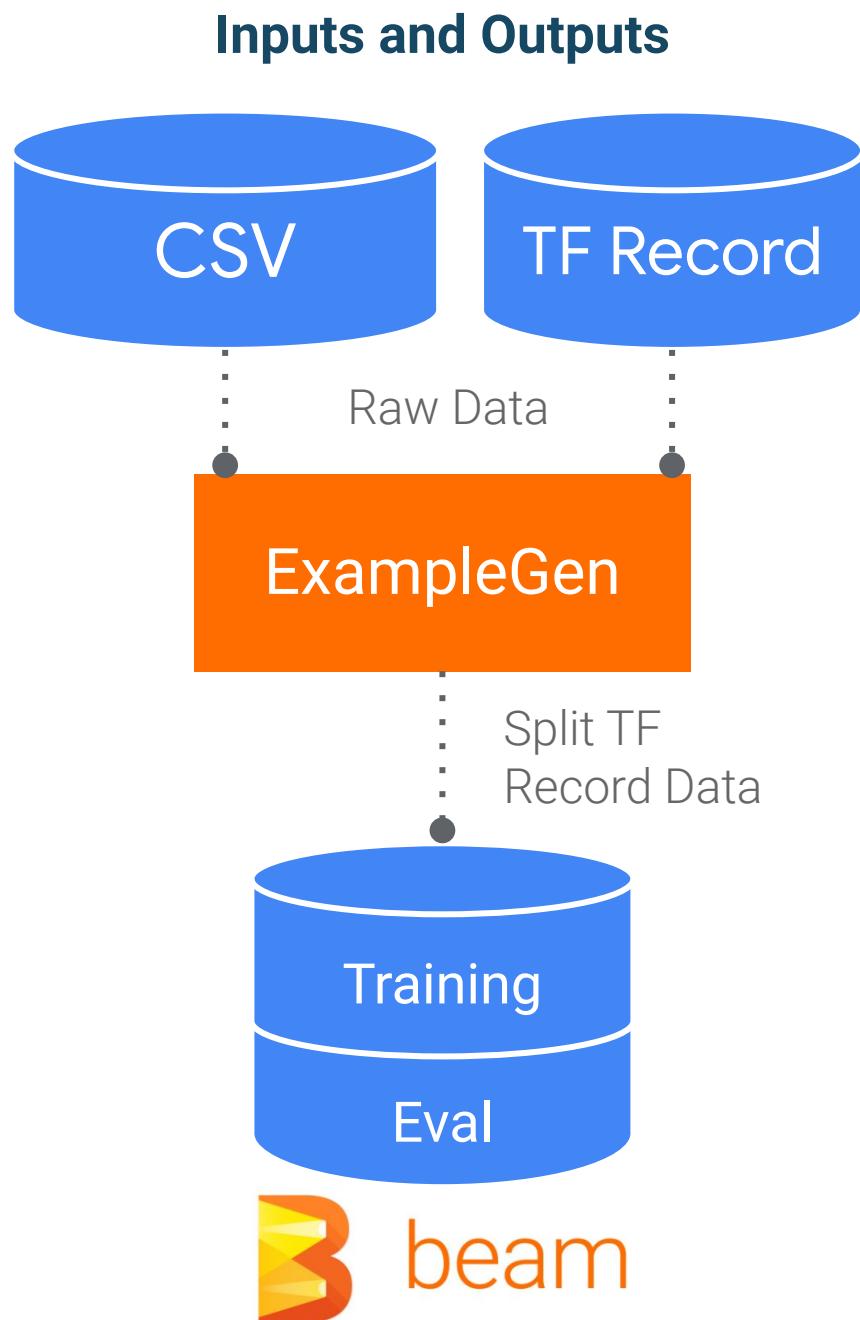
## Configuration

```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

## ML project lifecycle benefits

# Component: ExampleGen



## Configuration

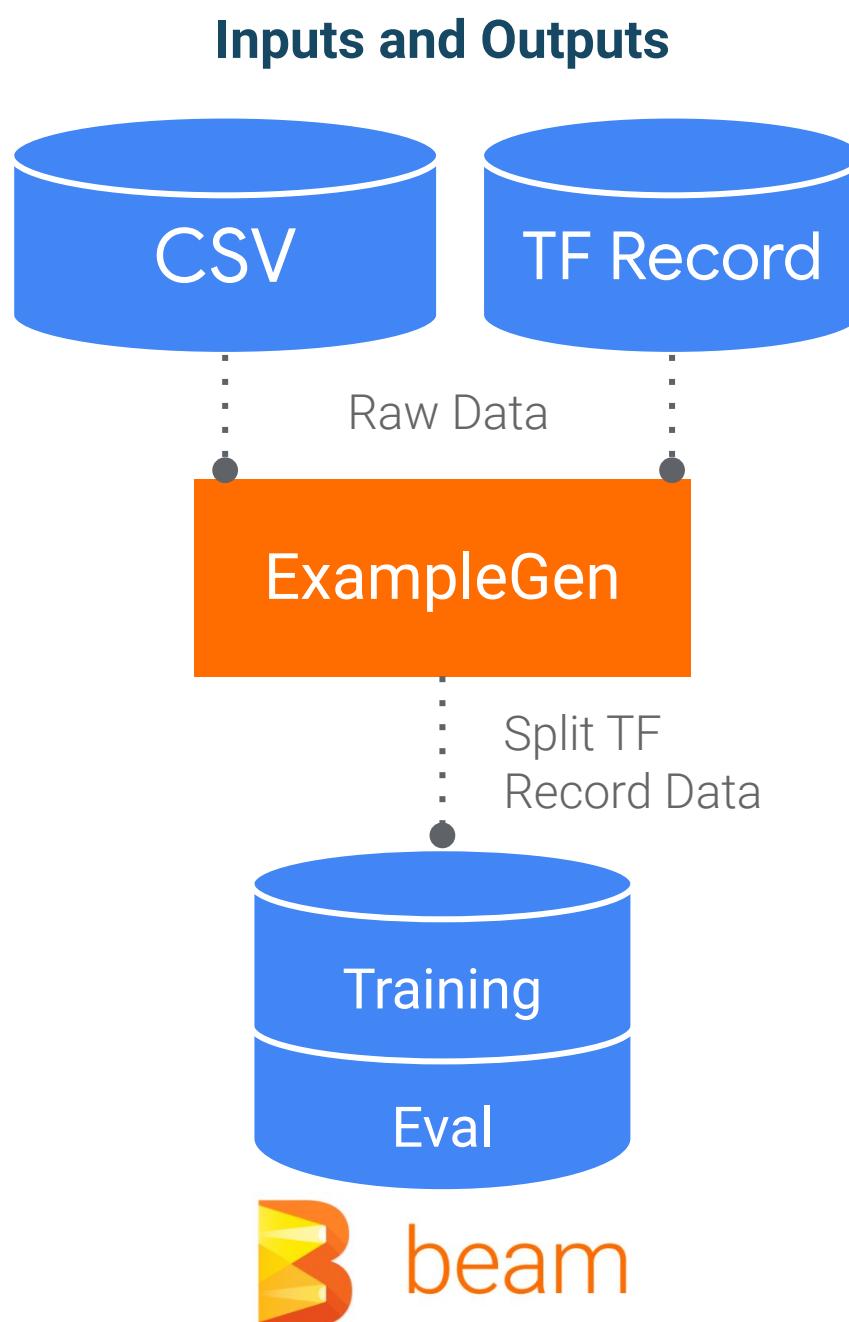
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

## ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling

# Component: ExampleGen



## Configuration

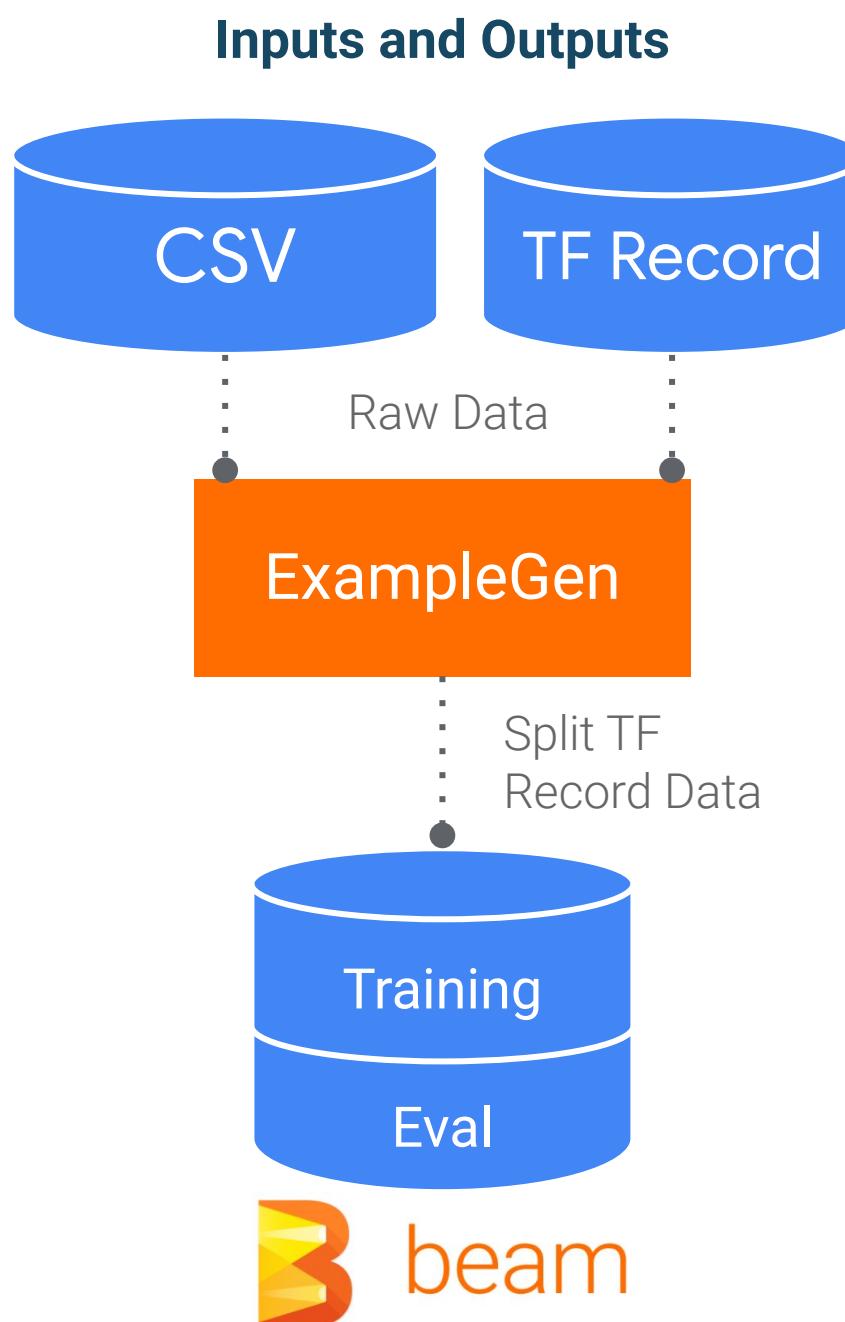
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

## ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion

# Component: ExampleGen



## Configuration

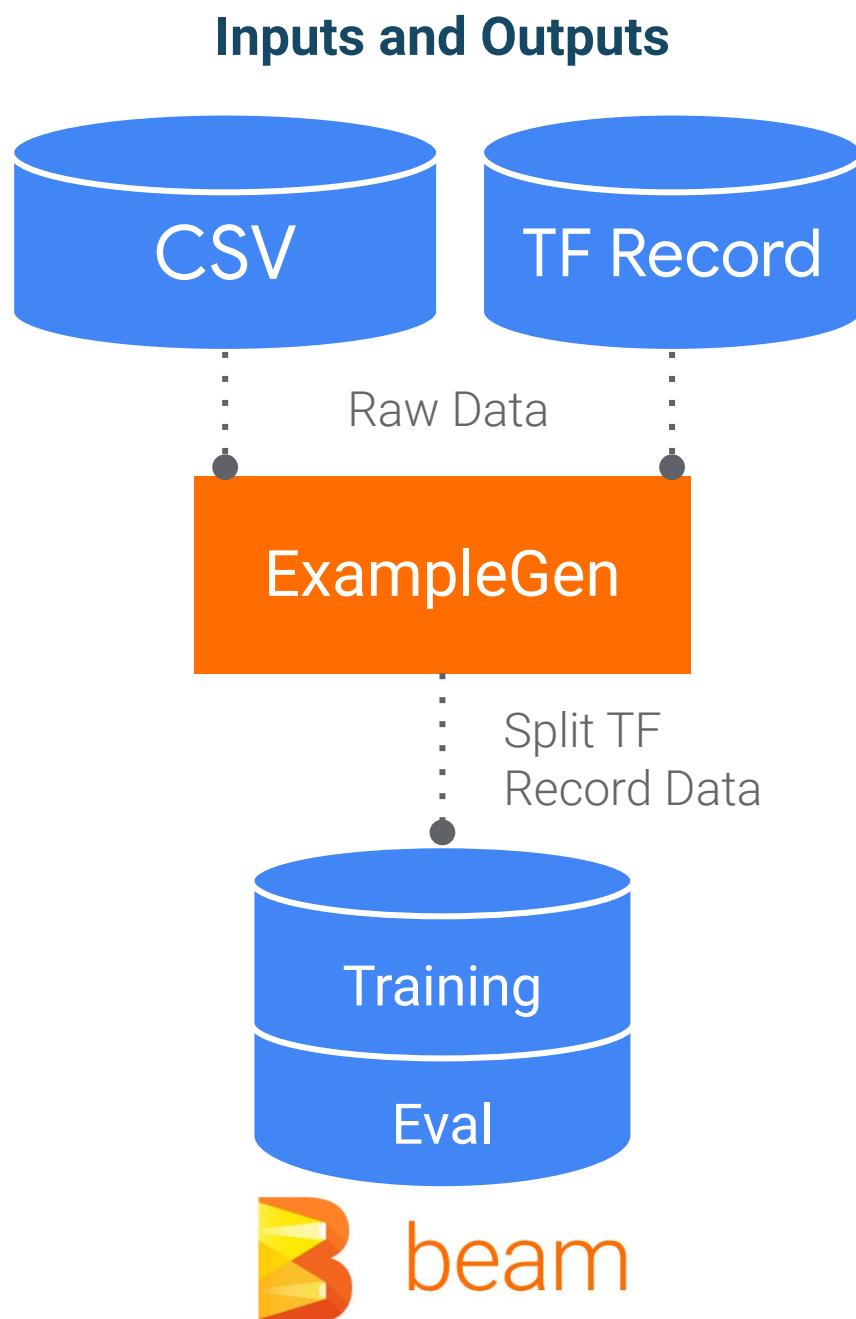
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

## ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion
- Apache Beam supported for scalable data ingestion

# Component: ExampleGen



## Configuration

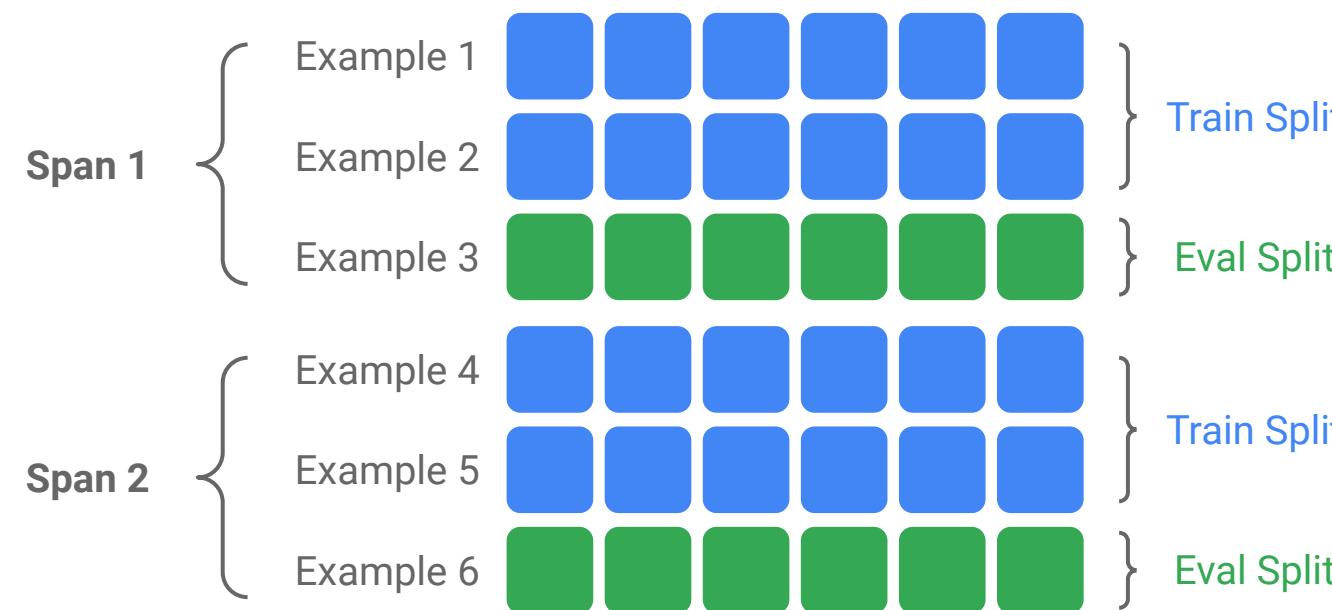
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

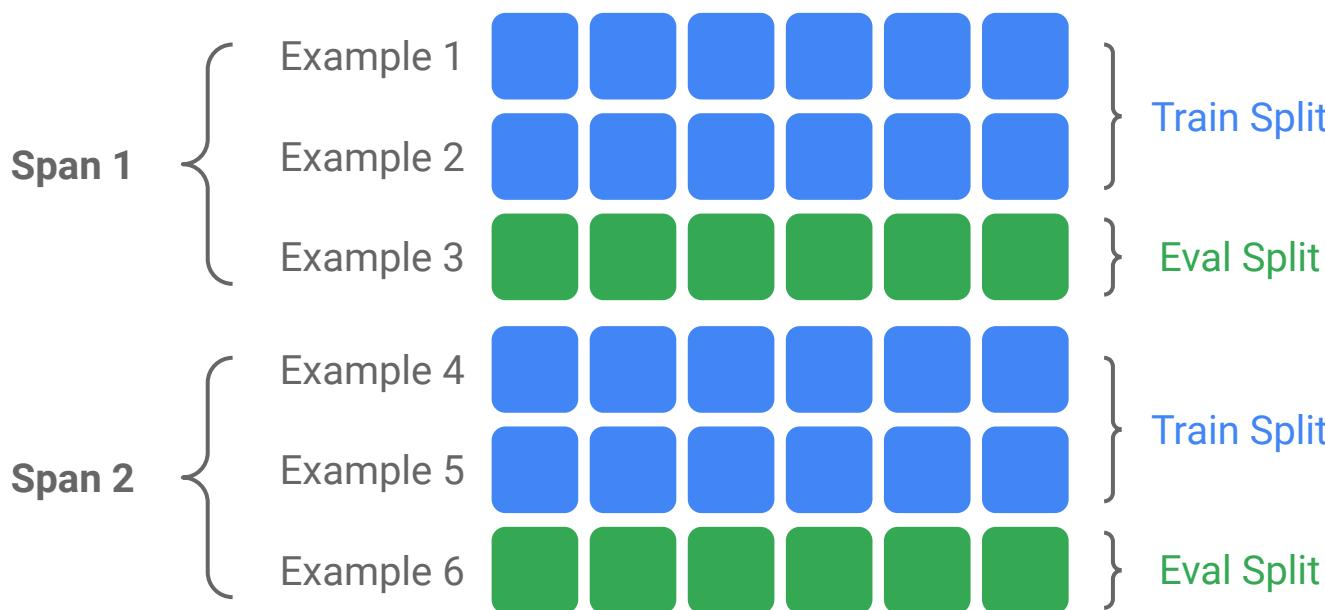
## ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion
- Apache Beam supported for scalable data ingestion
- Customizable to new input data formats and ingestion methods

# ExampleGen supports MLOps data management capabilities



# ExampleGen supports MLOps data management capabilities



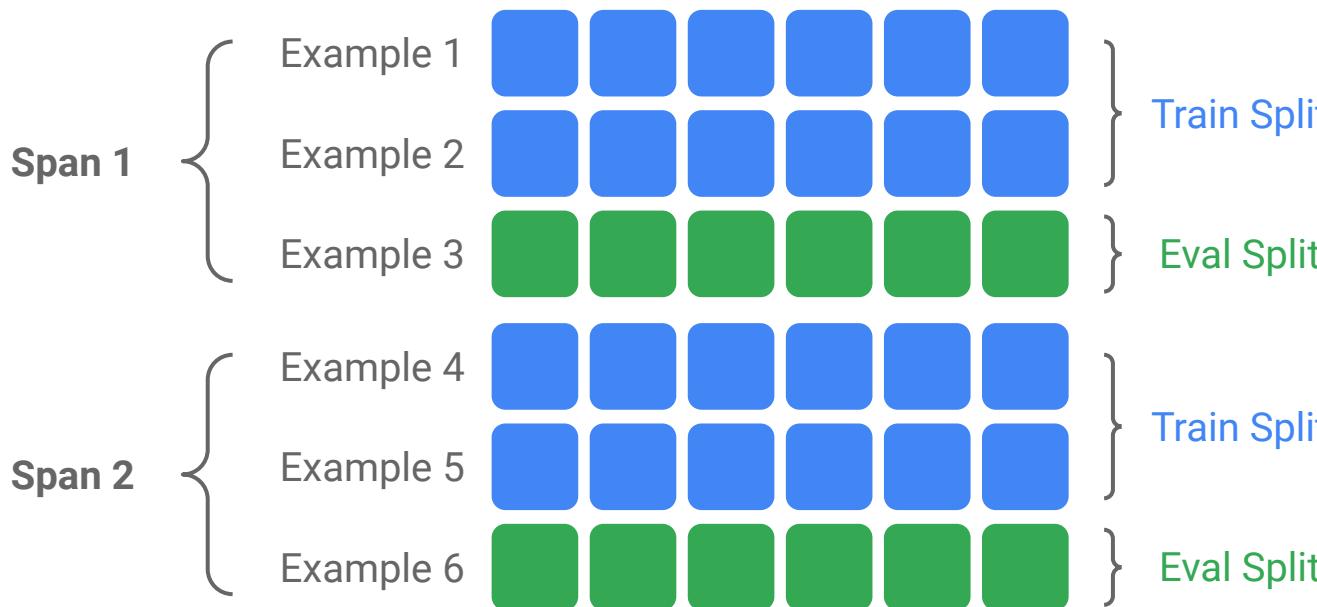
```
from tfx.proto import example_gen_pb2
from tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train',
                                 pattern='span-{SPAN:2}/train/*'),
    example_gen_pb2.Input.Split(name='eval',
                                 pattern='span-{SPAN:2}/eval/*'))]

range = range_config_pb2.RangeConfig(
    static_range=range_config_pb2.StaticRange(
        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                           range_config=range)
```

# ExampleGen supports MLOps data management capabilities



```
from tfx.proto import example_gen_pb2
from tfx.proto import range_config_pb2

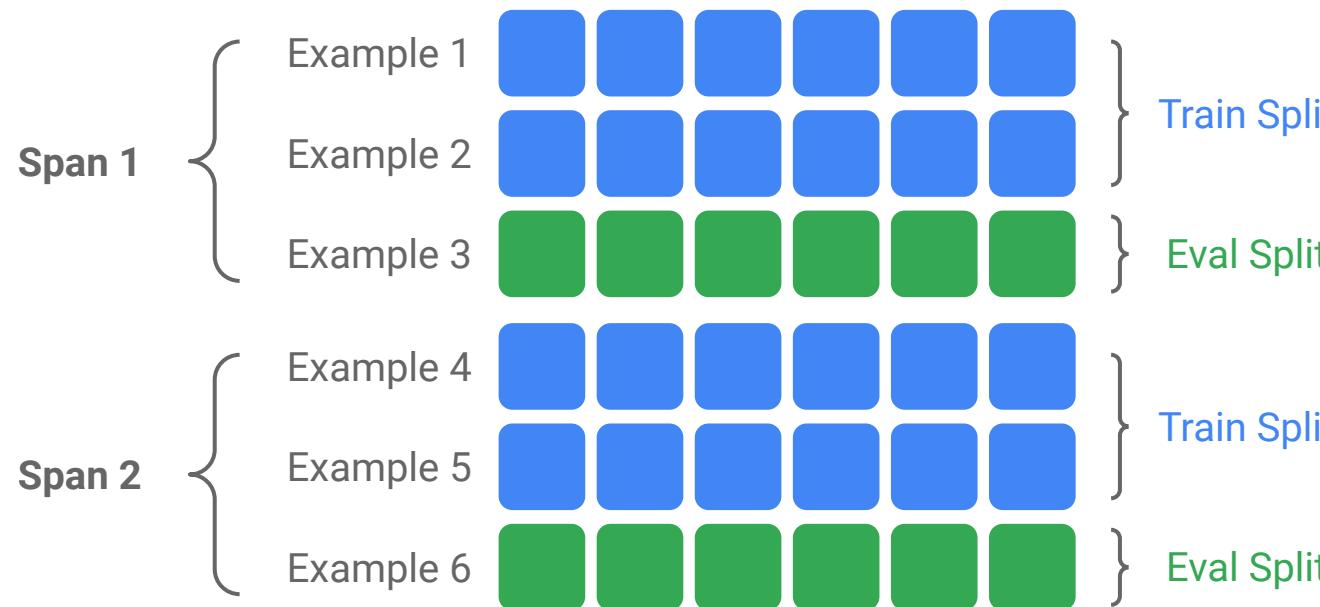
input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train',
                                 pattern='span-{SPAN:2}/train/*'),
    example_gen_pb2.Input.Split(name='eval',
                                 pattern='span-{SPAN:2}/eval/*'))]

range = range_config_pb2.RangeConfig(
    static_range=range_config_pb2.StaticRange(
        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                           range_config=range)
```

Customize data partitioning, spans, versioning, and raw data ingestion method through **ExampleGen Input and Output Configs**.

# ExampleGen supports MLOps data management capabilities



```
from tfx.proto import example_gen_pb2
from tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train',
                                 pattern='span-{SPAN:2}/train/*'),
    example_gen_pb2.Input.Split(name='eval',
                                 pattern='span-{SPAN:2}/eval/*'))]

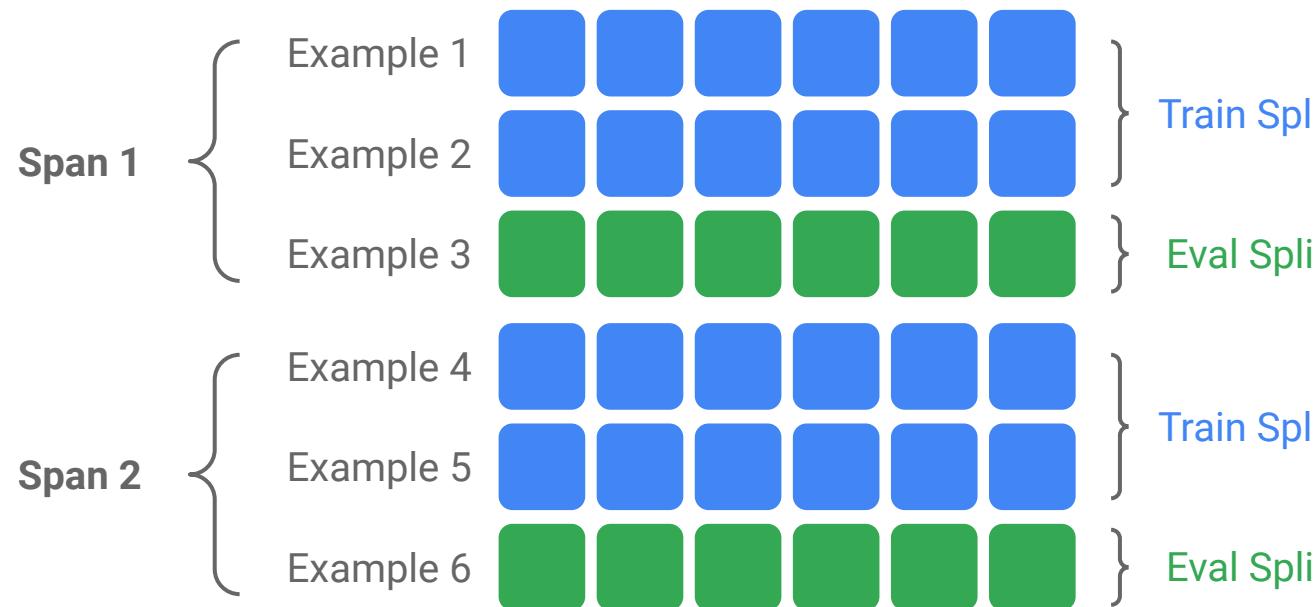
range = range_config_pb2.RangeConfig(
    static_range=range_config_pb2.StaticRange(
        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                           range_config=range)
```

Customize data partitioning, spans, versioning, and raw data ingestion method through **ExampleGen Input and Output Configs**.

Customize pipeline data spans with the **Range Config**.

# ExampleGen supports MLOps data management capabilities



```
from tfx.proto import example_gen_pb2
from tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train',
                                 pattern='span-{SPAN:2}/train/*'),
    example_gen_pb2.Input.Split(name='eval',
                                 pattern='span-{SPAN:2}/eval/*'))]

range = range_config_pb2.RangeConfig(
    static_range=range_config_pb2.StaticRange(
        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                           range_config=range)
```

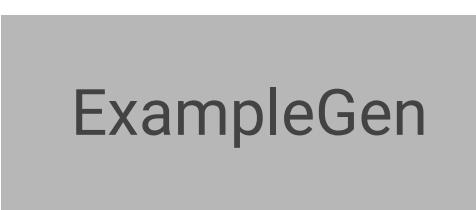
Customize data partitioning, spans, versioning, and raw data ingestion method through **ExampleGen Input and Output Configs**.

Customize pipeline data spans with the **Range Config**.

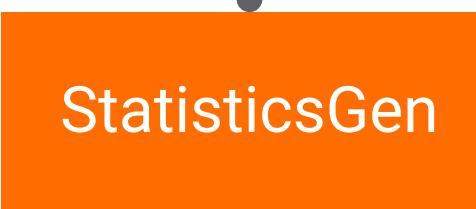
Support for external CSV, Avro, Parquet, BigQuery, and TFRecords.

# Component: StatisticsGen

## Inputs and Outputs



Data

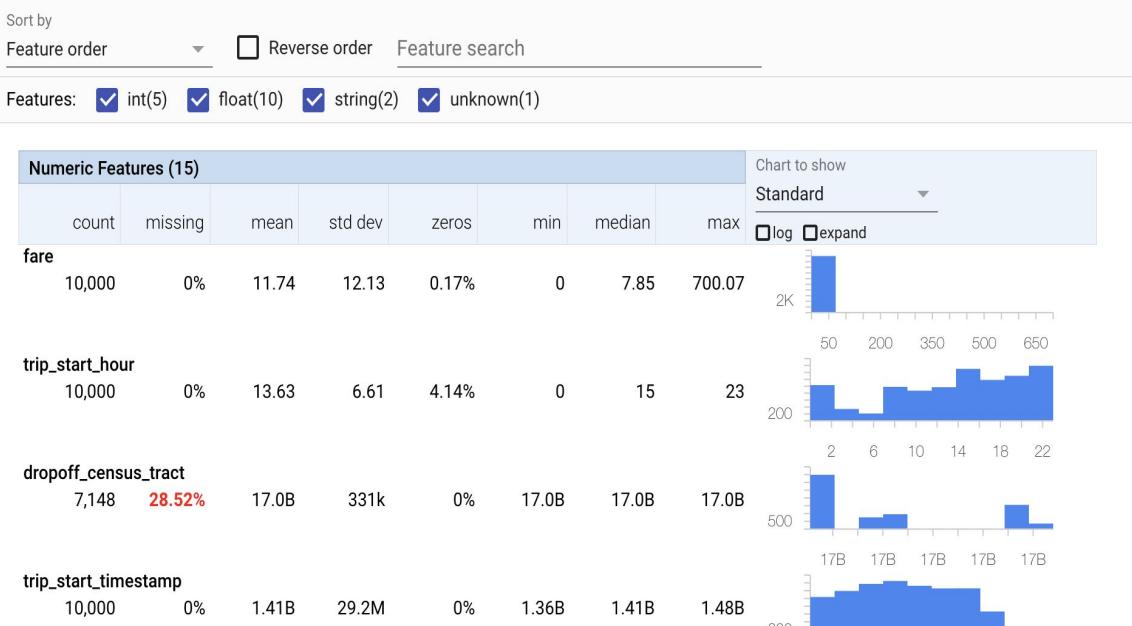


Statistics

## Configuration

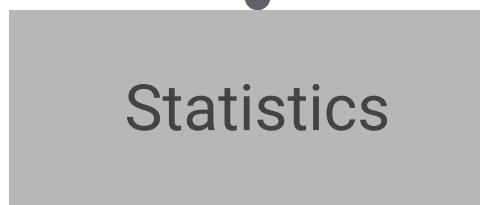
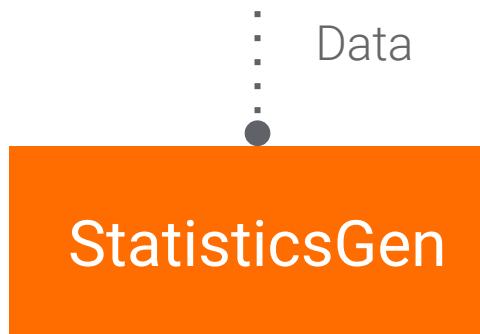
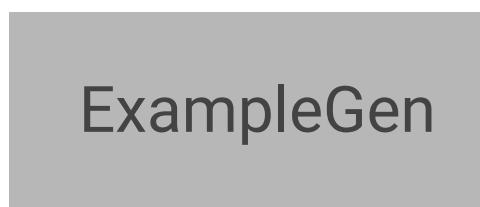
```
statistics_gen = tfx.components.StatisticsGen(  
    instance_name='Statistics_Generation',  
    examples=example_gen.outputs['examples'])
```

## TFDV Feature Statistics Visualization



# Component: StatisticsGen

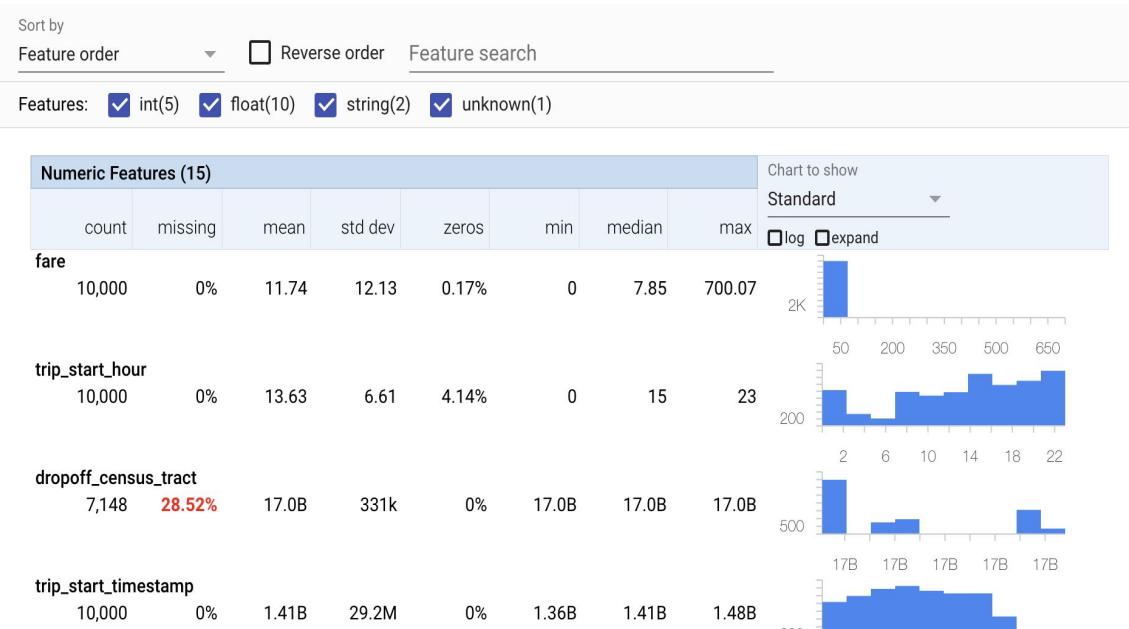
## Inputs and Outputs



## Configuration

```
statistics_gen = tfx.components.StatisticsGen(  
    instance_name='Statistics_Generation',  
    examples=example_gen.outputs['examples'])
```

## TFDV Feature Statistics Visualization

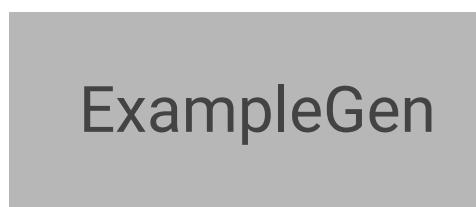


## ML project lifecycle benefits



# Component: StatisticsGen

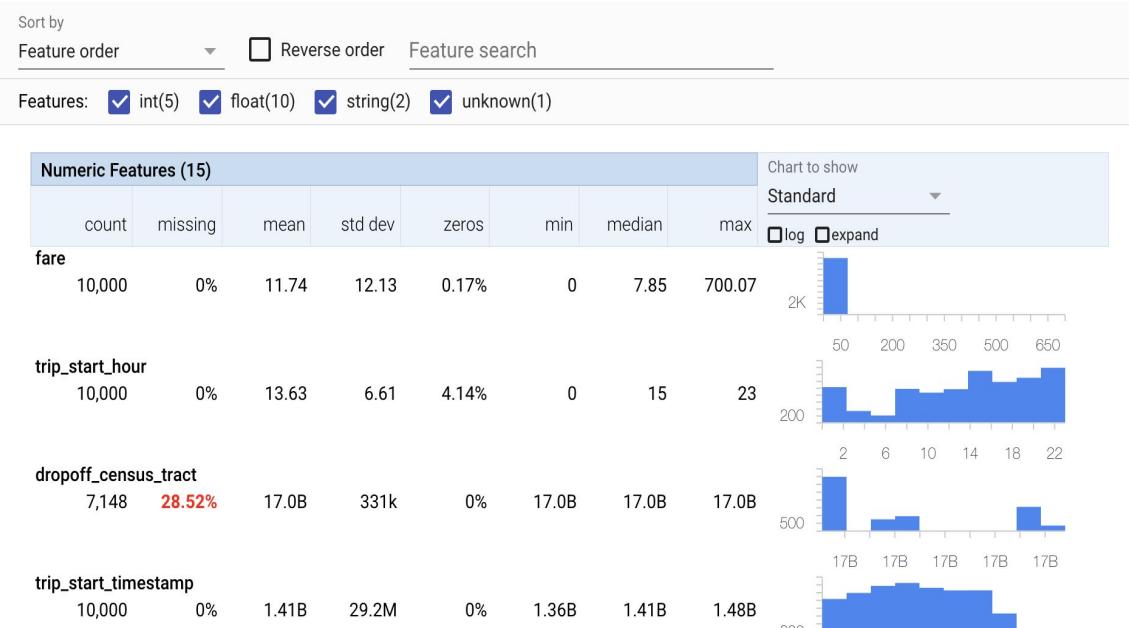
## Inputs and Outputs



## Configuration

```
statistics_gen = tfx.components.StatisticsGen(  
    instance_name='Statistics_Generation',  
    examples=example_gen.outputs['examples'])
```

## TFDV Feature Statistics Visualization

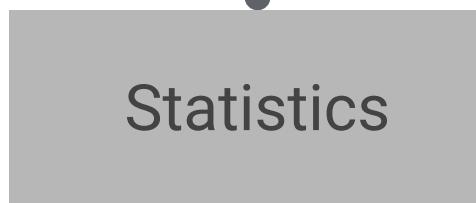
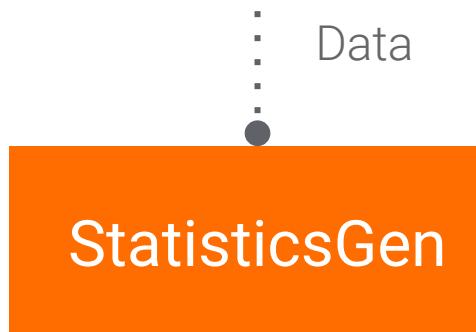
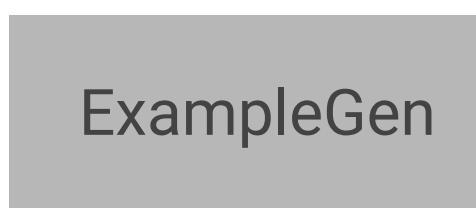


## ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) library for calculating feature statistics

# Component: StatisticsGen

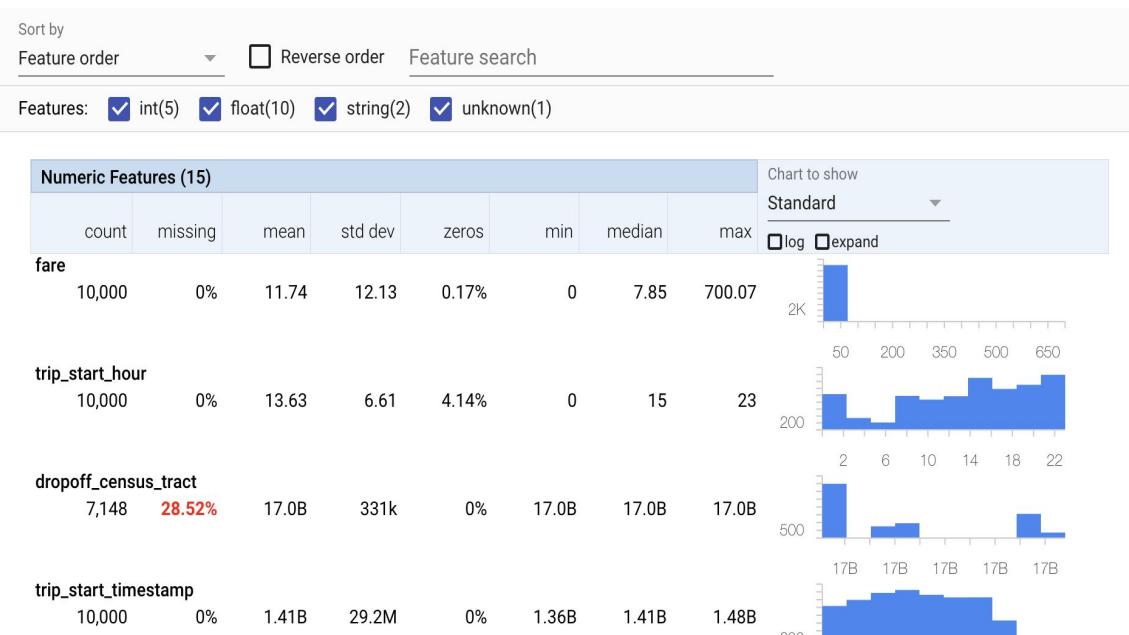
## Inputs and Outputs



## Configuration

```
statistics_gen = tfx.components.StatisticsGen(  
    instance_name='Statistics_Generation',  
    examples=example_gen.outputs['examples'])
```

## TFDV Feature Statistics Visualization



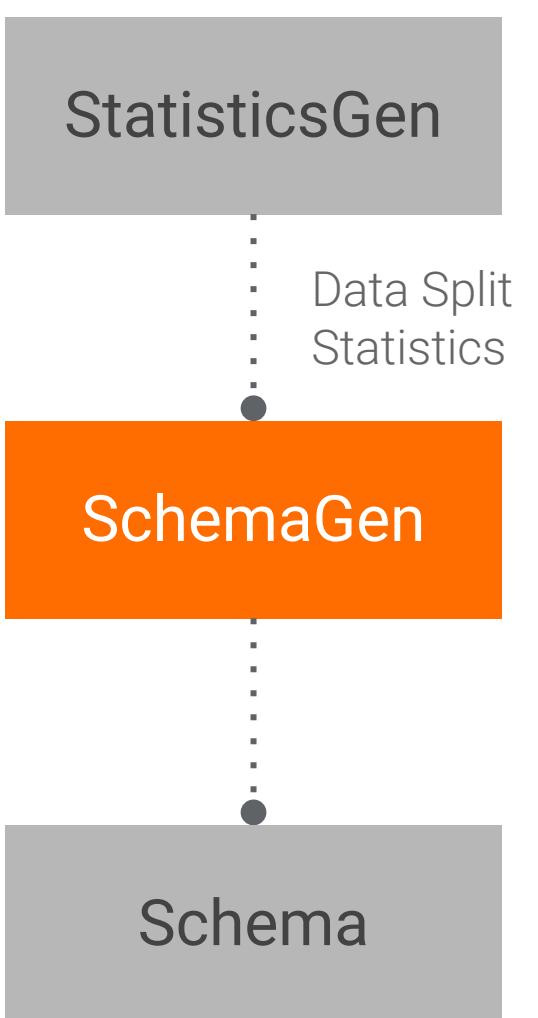
## ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) library for calculating feature statistics
- Scalable full-pass dataset feature statistics processing with Apache Beam

---

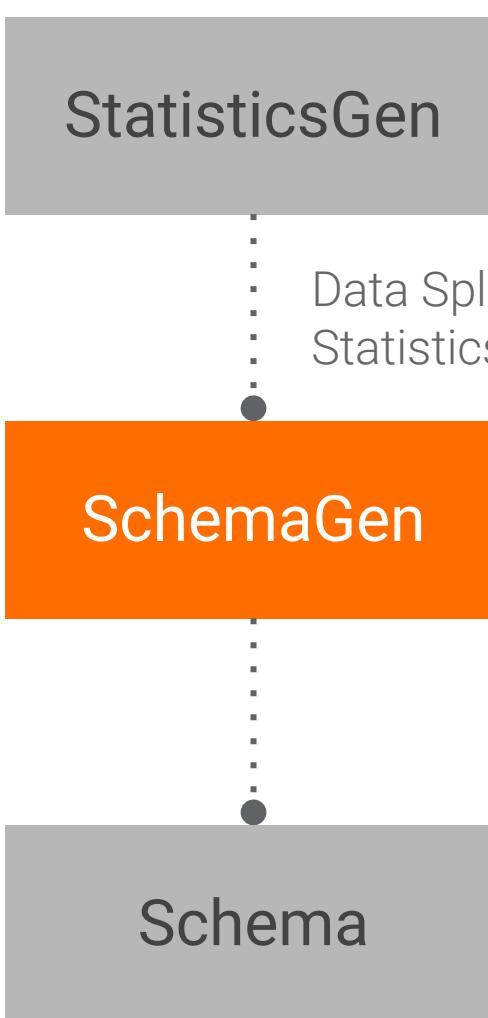
# Component: SchemaGen

## Inputs and Outputs



# Component: SchemaGen

## Inputs and Outputs



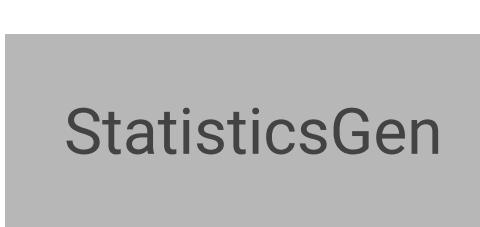
## Configuration

```
schema_gen = SchemaGen(  
    statistics=statistics_gen.outputs['statistics'],  
    infer_feature_shape=False)
```

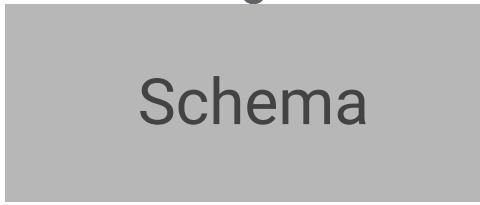


# Component: SchemaGen

## Inputs and Outputs



Data Split  
Statistics



## Configuration

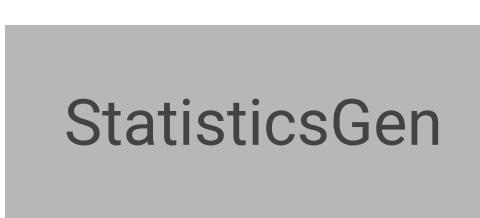
```
schema_gen = SchemaGen(  
    statistics=statistics_gen.outputs['statistics'],  
    infer_feature_shape=False)
```

Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional	-	-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

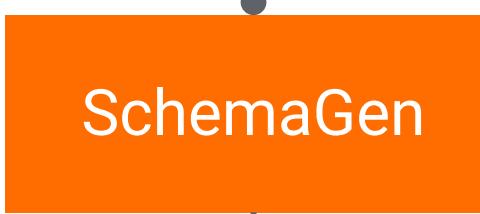


# Component: SchemaGen

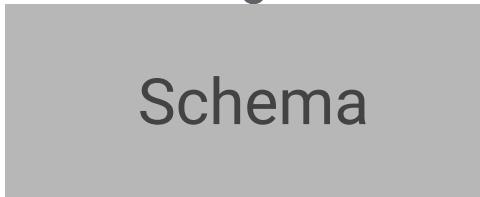
## Inputs and Outputs



Data Split  
Statistics



StatisticsGen



SchemaGen

Schema



## Configuration

```
schema_gen = SchemaGen(  
    statistics=statistics_gen.outputs['statistics'],  
    infer_feature_shape=False)
```

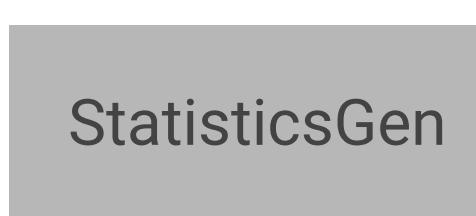
Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional	-	-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

## ML project lifecycle benefits

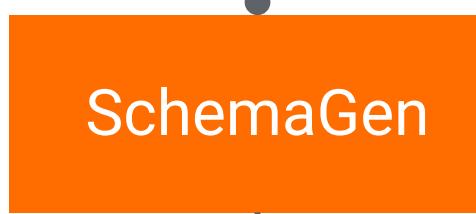
- TensorFlow Data Validation (TFDV) creates common data description for pipeline components

# Component: SchemaGen

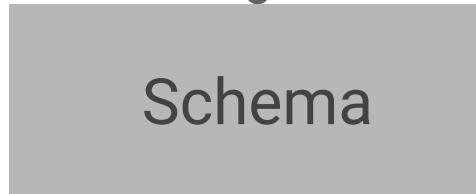
## Inputs and Outputs



Data Split  
Statistics



StatisticsGen



SchemaGen

Schema



## Configuration

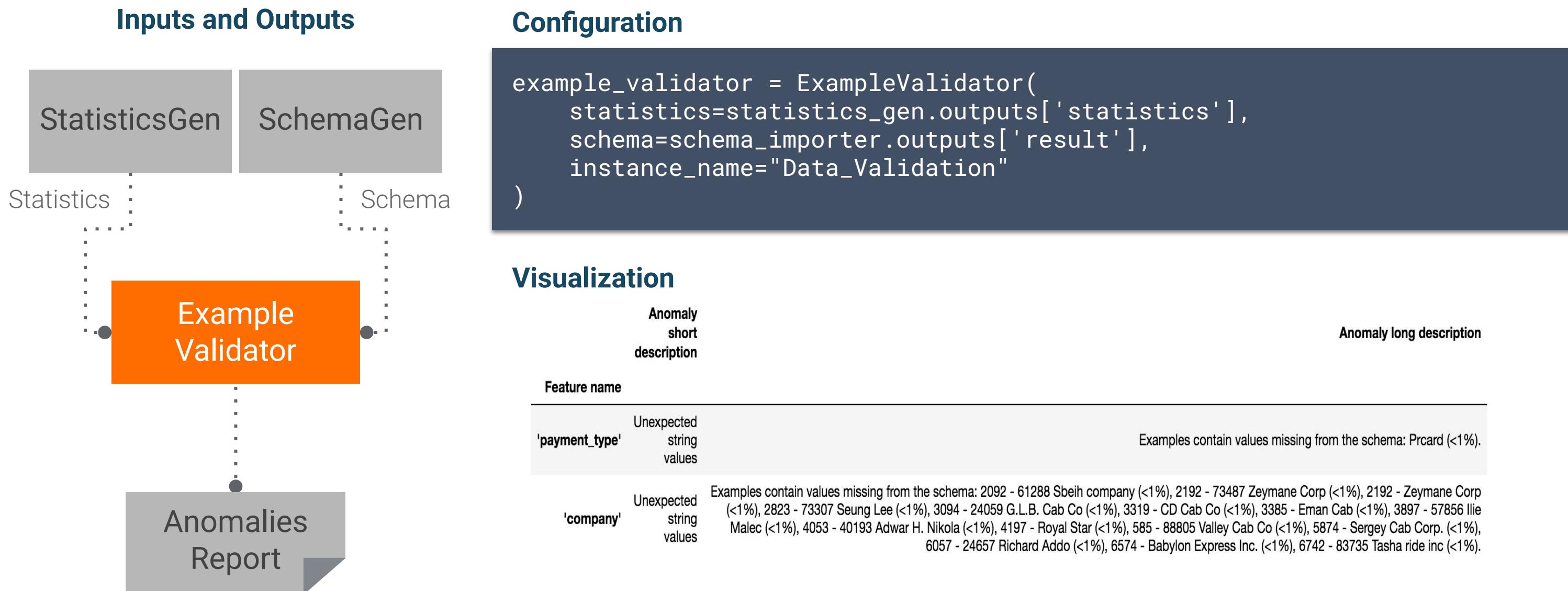
```
schema_gen = SchemaGen(  
    statistics=statistics_gen.outputs['statistics'],  
    infer_feature_shape=False)
```

Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional	-	-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

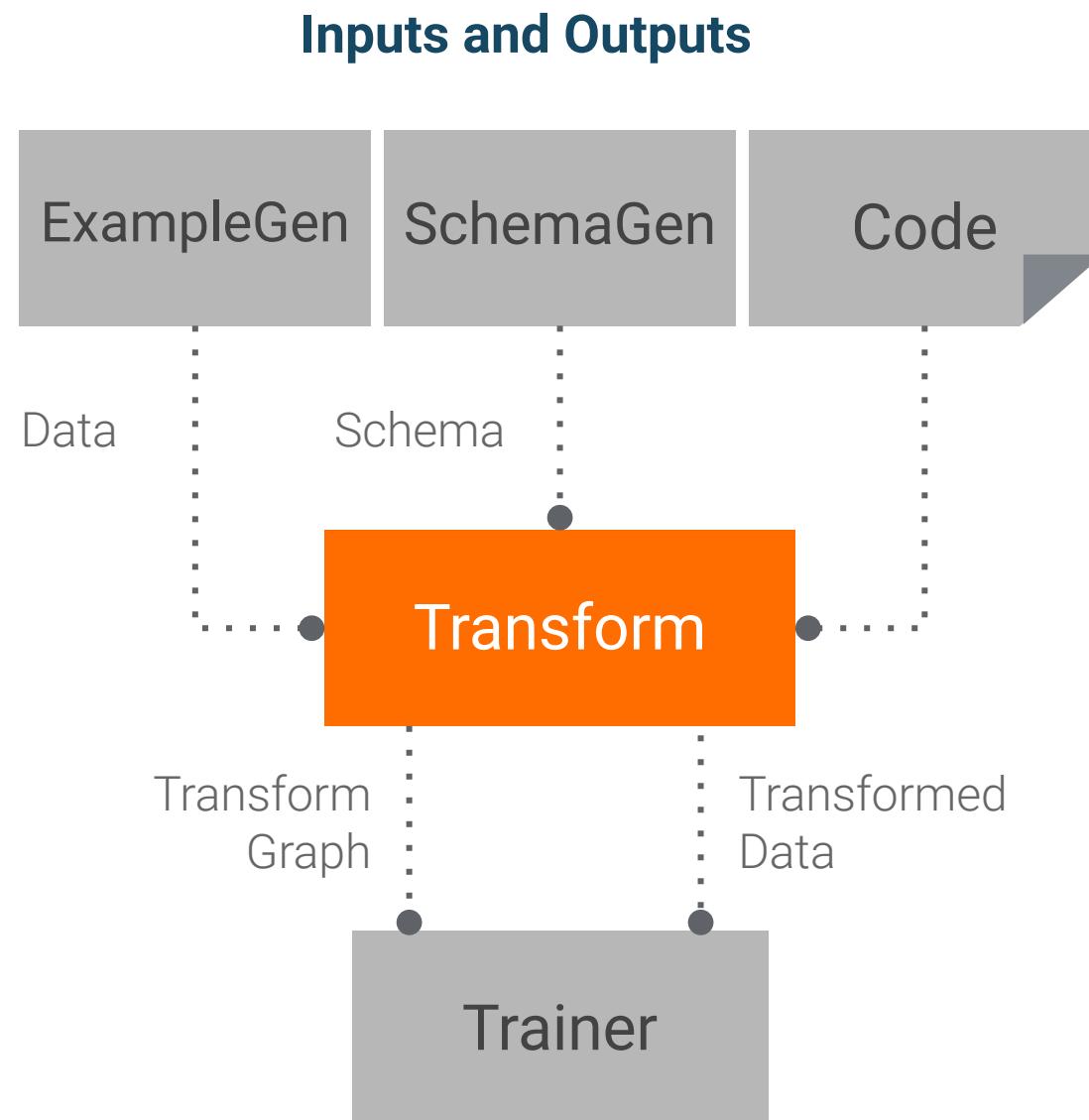
## ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) creates common data description for pipeline components
- A Schema enables continuous data monitoring and validation during pipeline training

# Component: ExampleValidator



# Component: Transform

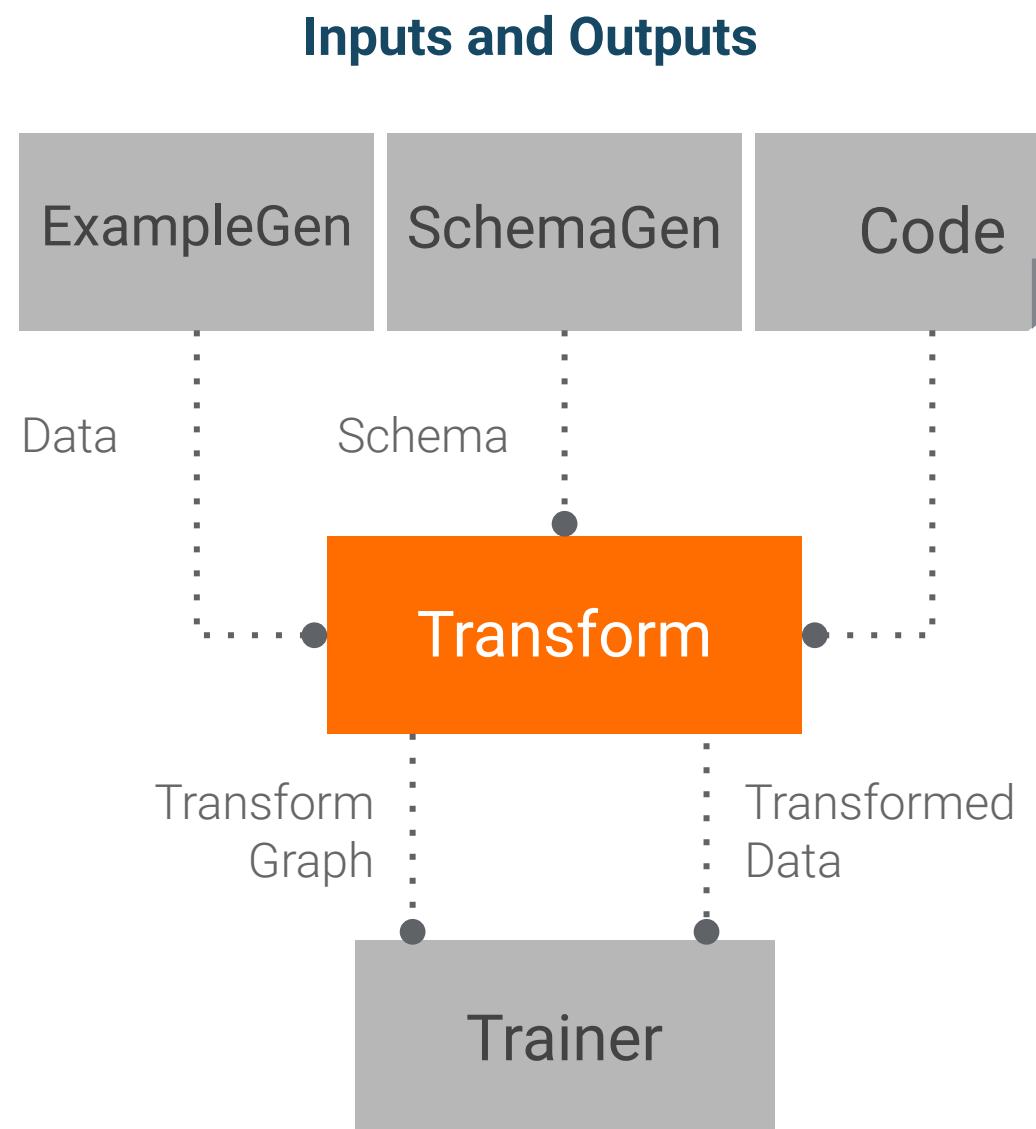


## Configuration

```
transform = Transform(  
    examples=example_gen.outputs['examples'],  
    schema=schema_importer.outputs['result'],  
    module_file=TRANSFORM_MODULE)
```



# Component: Transform



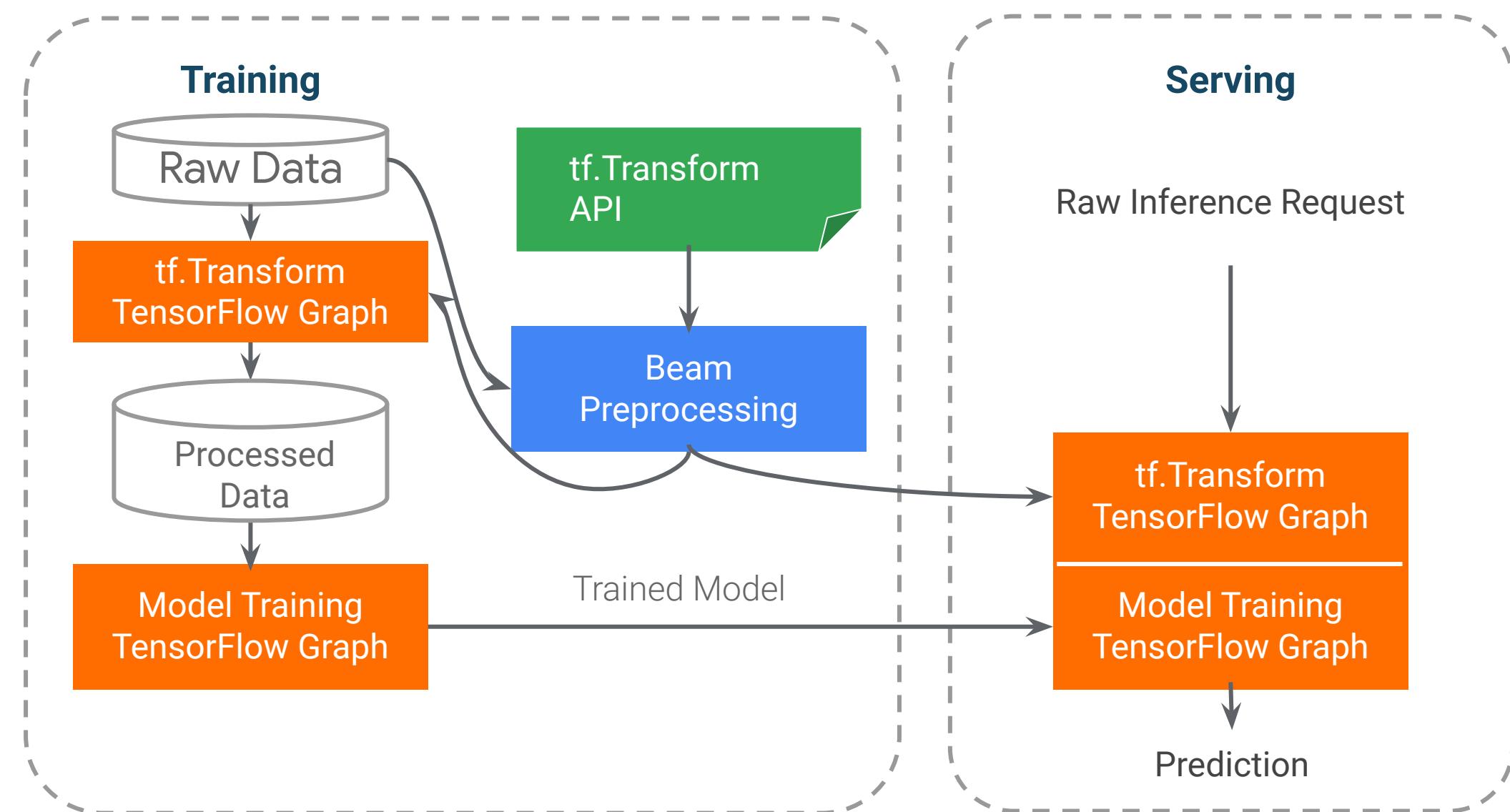
## Configuration

```
transform = Transform(  
    examples=example_gen.outputs['examples'],  
    schema=schema_importer.outputs['result'],  
    module_file=TRANSFORM_MODULE)
```

## Tensorflow Transform (TFT) library code in pipeline/preprocessing.py

```
def preprocessing_fn(inputs):  
  
    outputs = {}  
  
    # Scale numerical features  
    for key in features.NUMERIC_FEATURE_KEYS:  
        outputs[features.transformed_name(key)] = tft.scale_to_z_score(  
            _fill_in_missing(inputs[key]))  
  
    # Generate vocabularies and maps categorical features  
    for key in features.CATEGORICAL_FEATURE_KEYS:  
        outputs[features.transformed_name(key)] = tft.compute_and_apply_vocabulary(  
            x=_fill_in_missing(inputs[key]), num_oov_buckets=1, vocab_filename=key)  
  
    outputs[features.transformed_name(features.LABEL_KEY)] = _fill_in_missing(  
        inputs[features.LABEL_KEY])  
  
    return outputs
```

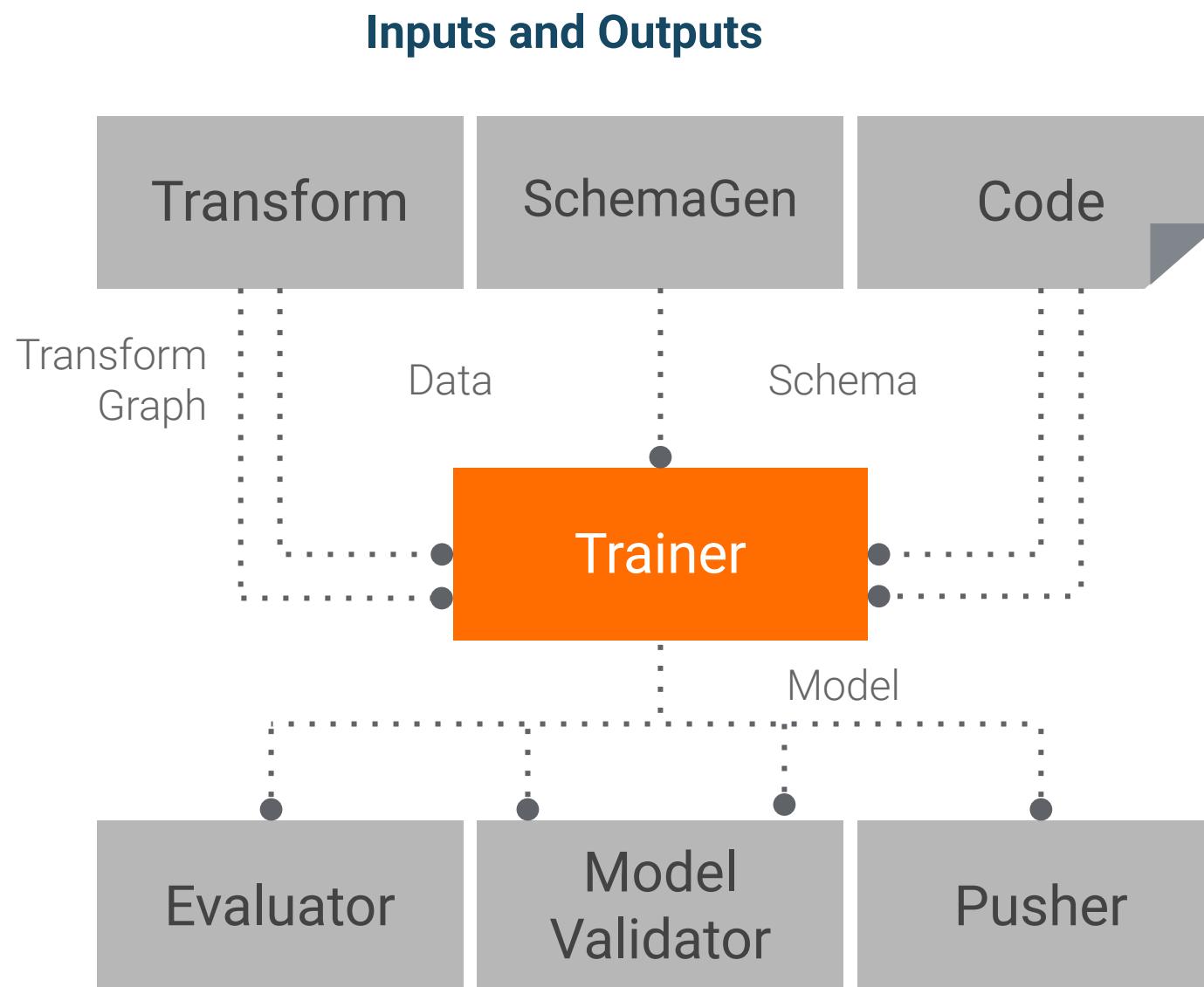
# Component: Transform



## ML project lifecycle benefits

- Generates SavedModel for consistent training and serving feature engineering
- Backed by Apache Beam for scalable distributed data processing to large datasets

# Component: Trainer



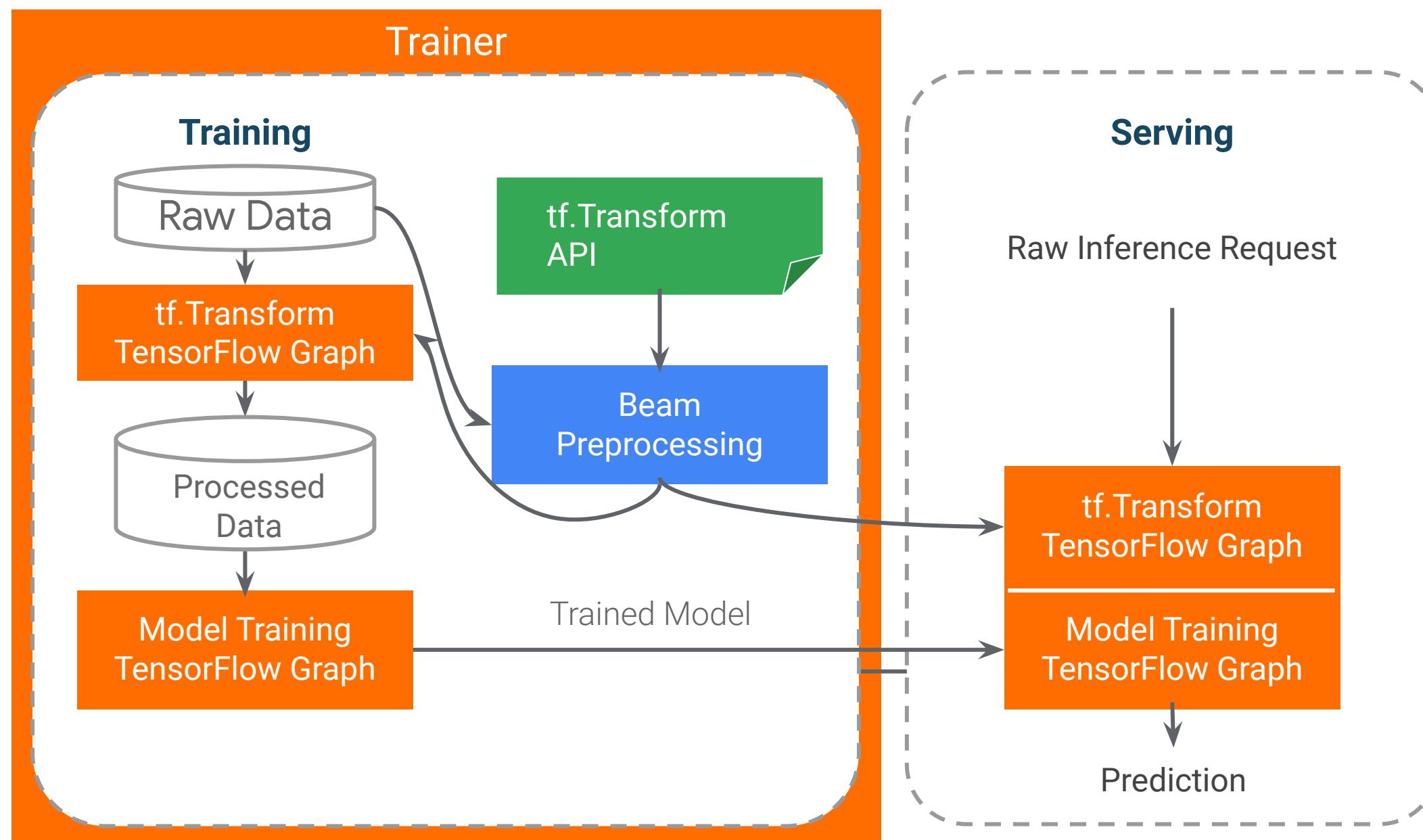
## Configuration

```
trainer = Trainer(  
    custom_executor_spec=executor_spec.ExecutorClassSpec(trainer_executor.GenericExecutor),  
    module_file=model.py,  
    transformed_examples=transform.outputs["transformed_examples"],  
    schema=schema_importer.outputs["result"],  
    transform_graph=transform.outputs["transform_graph"],  
    train_args=trainer_pb2.TrainArgs(num_steps=5000),  
    eval_args=trainer_pb2.EvalArgs(num_steps=1000))
```

**Code -> your TensorFlow code in a pipeline/models/model.py**



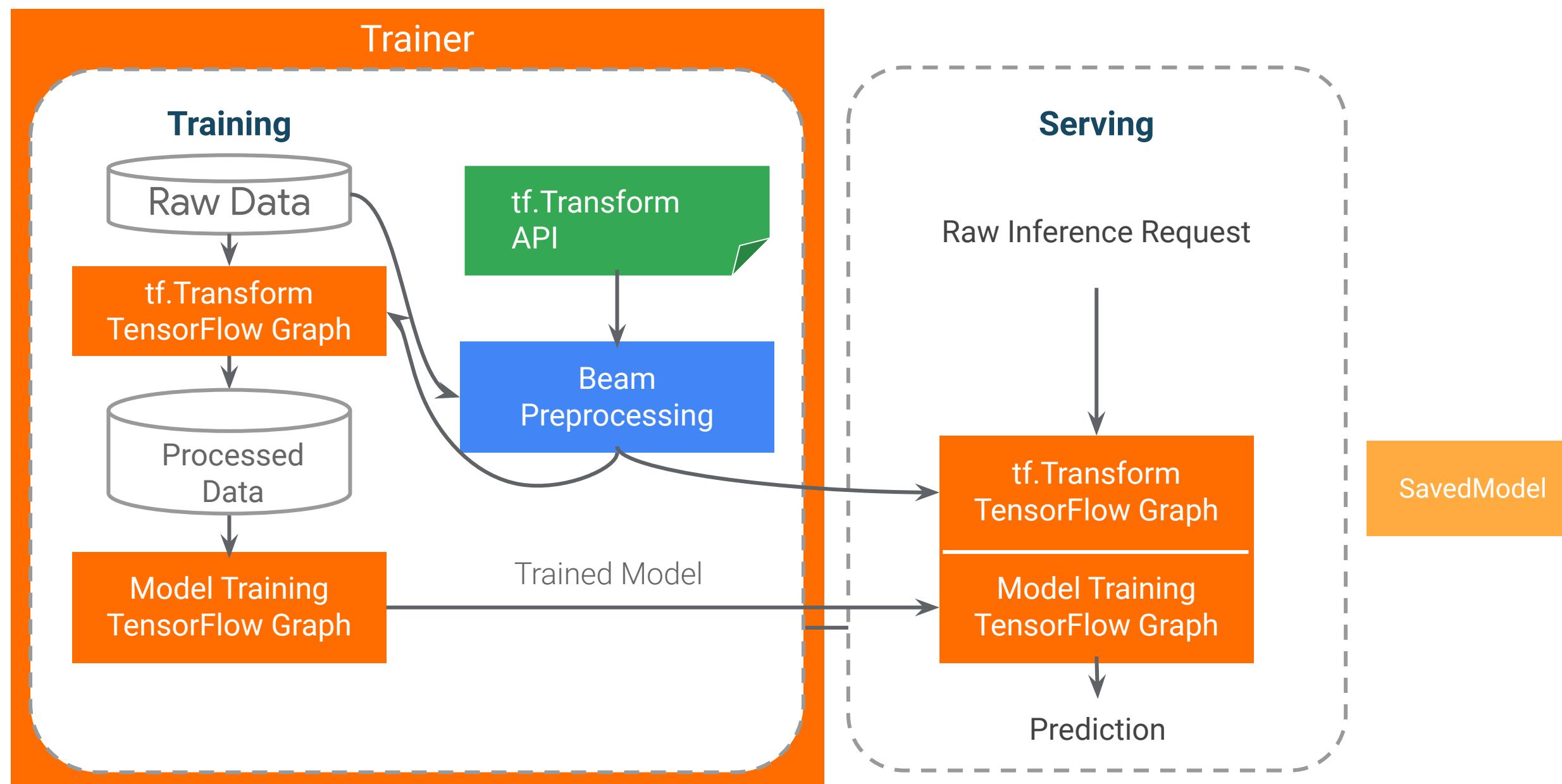
# Component: Trainer



## ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

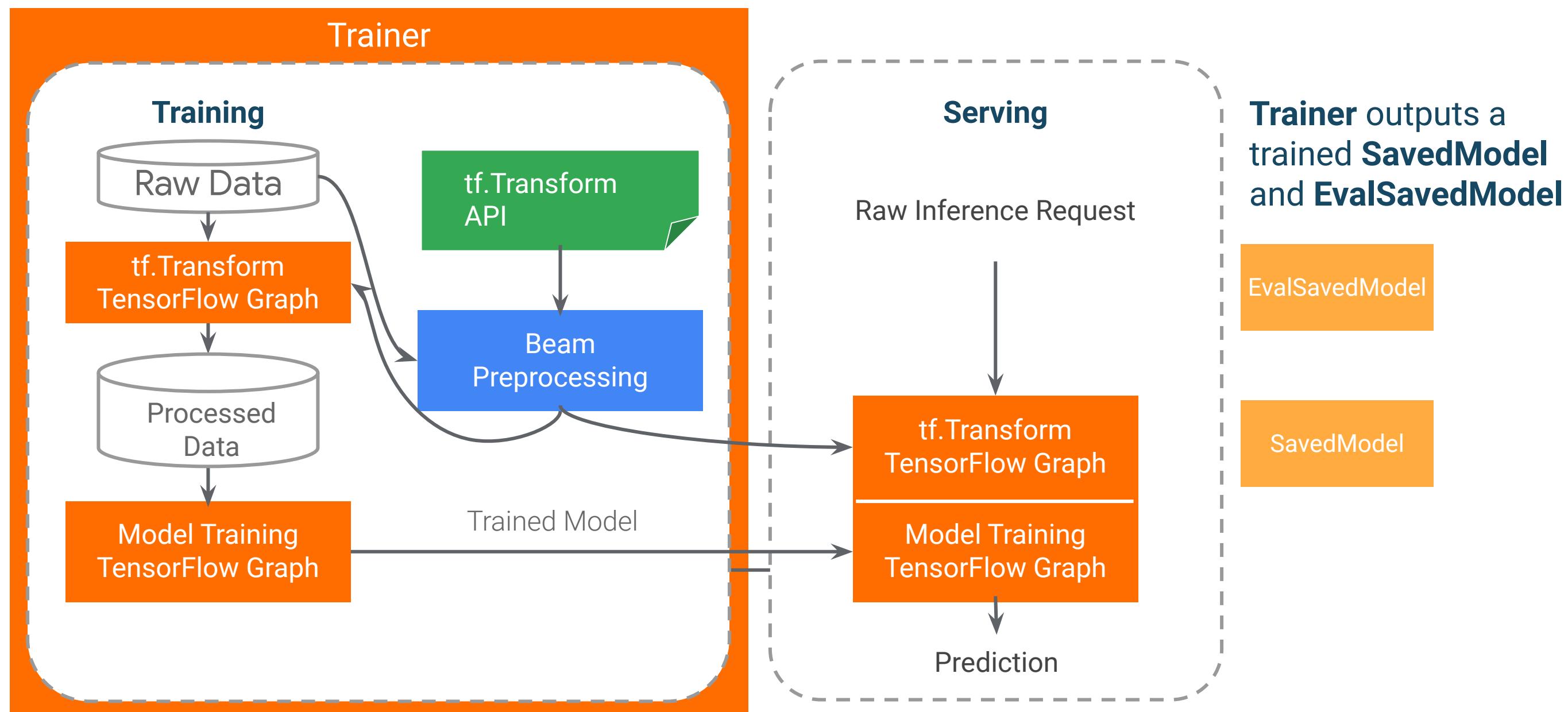
# Component: Trainer



## ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

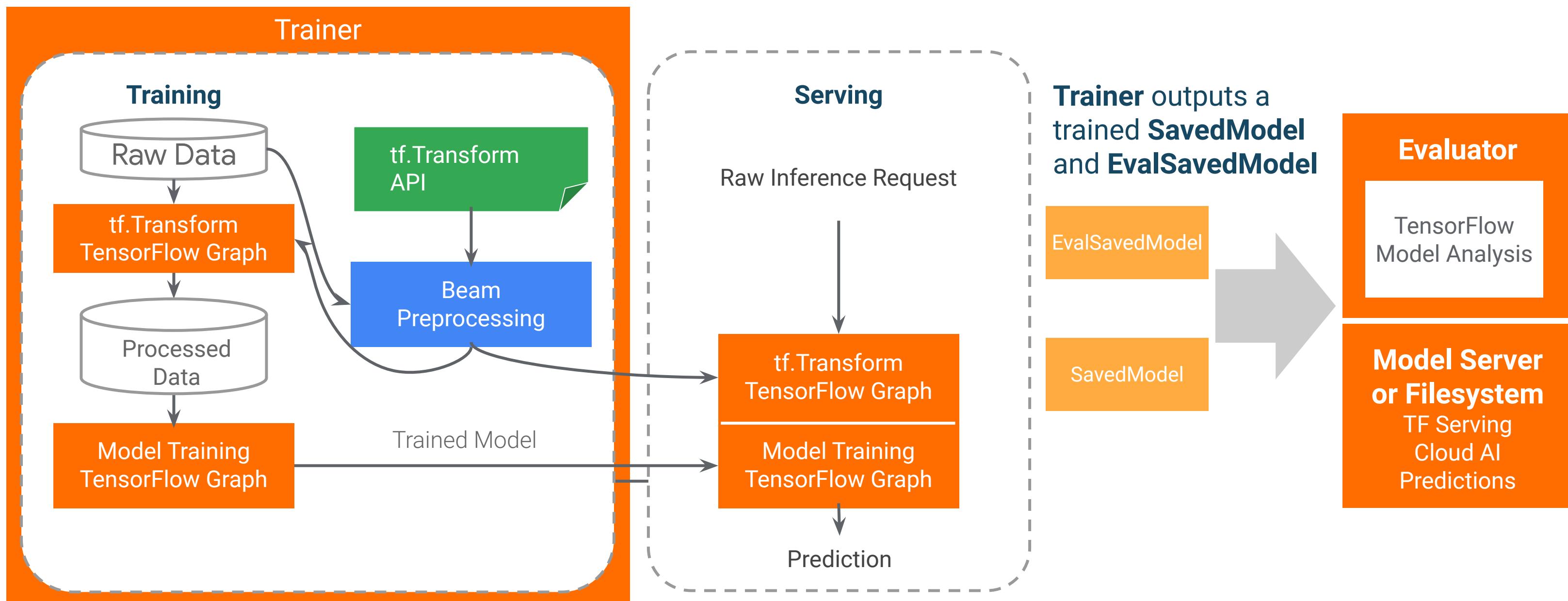
# Component: Trainer



## ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

# Component: Trainer

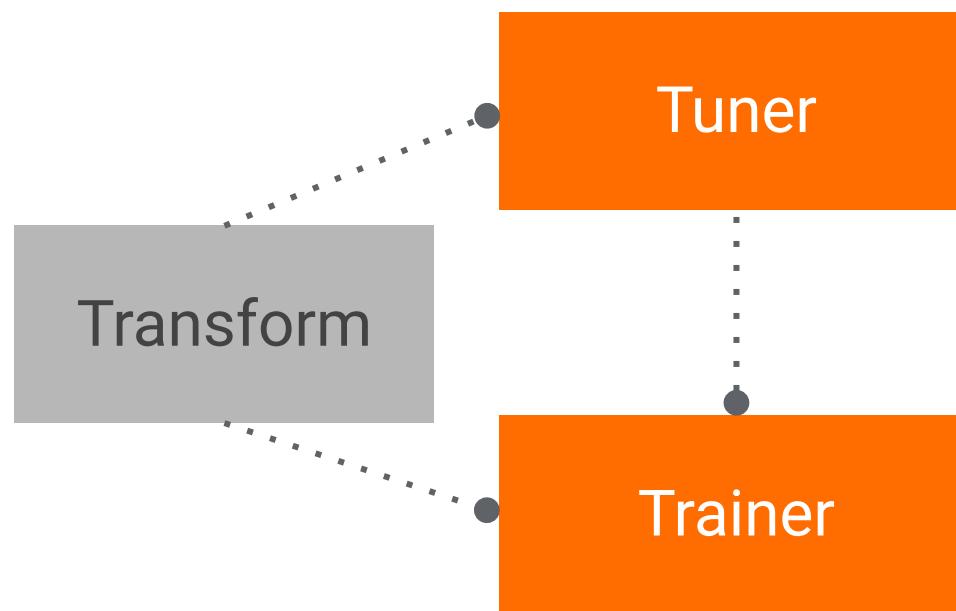


## ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

# Component: Tuner

## Tuner works with Trainer

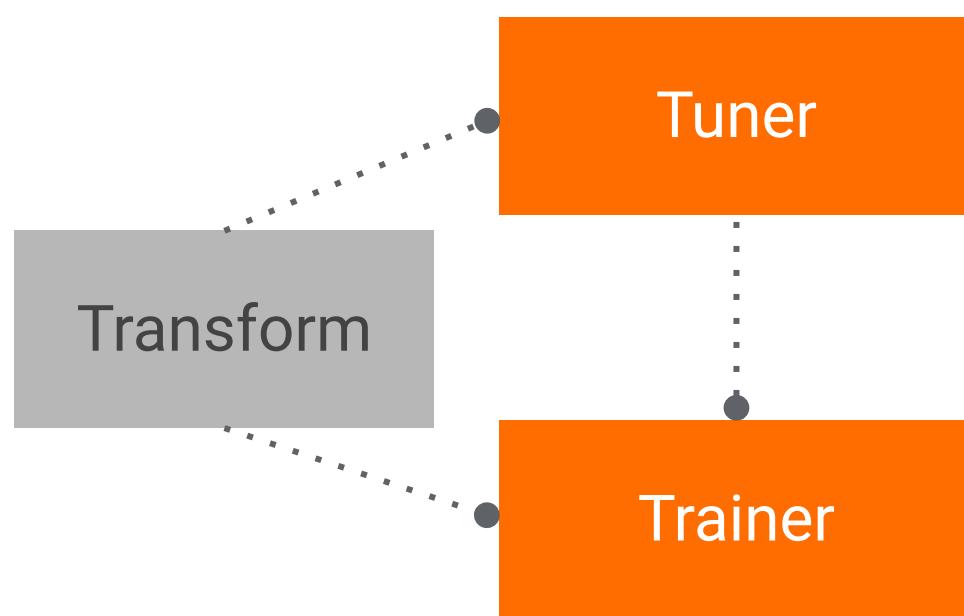


## Configuration

```
tuner = Tuner(  
    module_file=module_file, # Contains `tuner_fn`.  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=20),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

# Component: Tuner

## Tuner works with Trainer



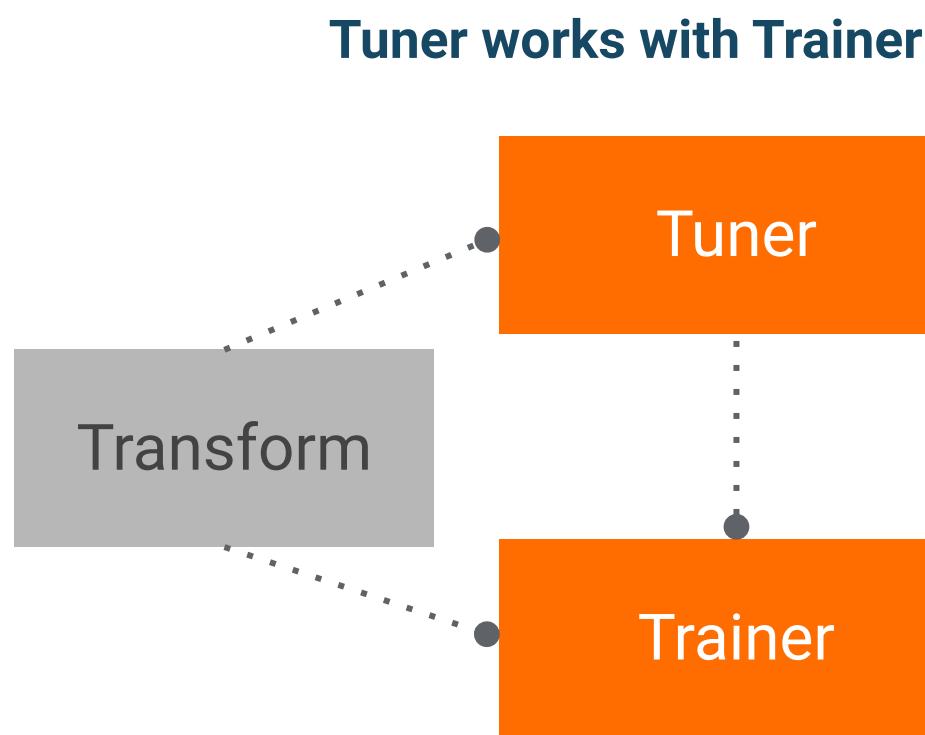
## Configuration

```
tuner = Tuner(  
    module_file=module_file, # Contains `tuner_fn`.  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=20),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

## Best hyperparameters added to Trainer in pipeline

```
trainer = Trainer(  
    module_file=module_file, # Contains `run_fn`.  
    custom_executor_spec=executor_spec,  
    ExecutorClassSpec(GenericExecutor),  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    schema=schema_gen.outputs['schema'],  
    # This will be passed to `run_fn`.  
    hyperparameters=tuner.outputs['best_hyperparameters'],  
    train_args=trainer_pb2.TrainArgs(num_steps=100),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

# Component: Tuner



## Configuration

```
tuner = Tuner(  
    module_file=module_file, # Contains `tuner_fn`.  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=20),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

## Best hyperparameters added to Trainer in pipeline

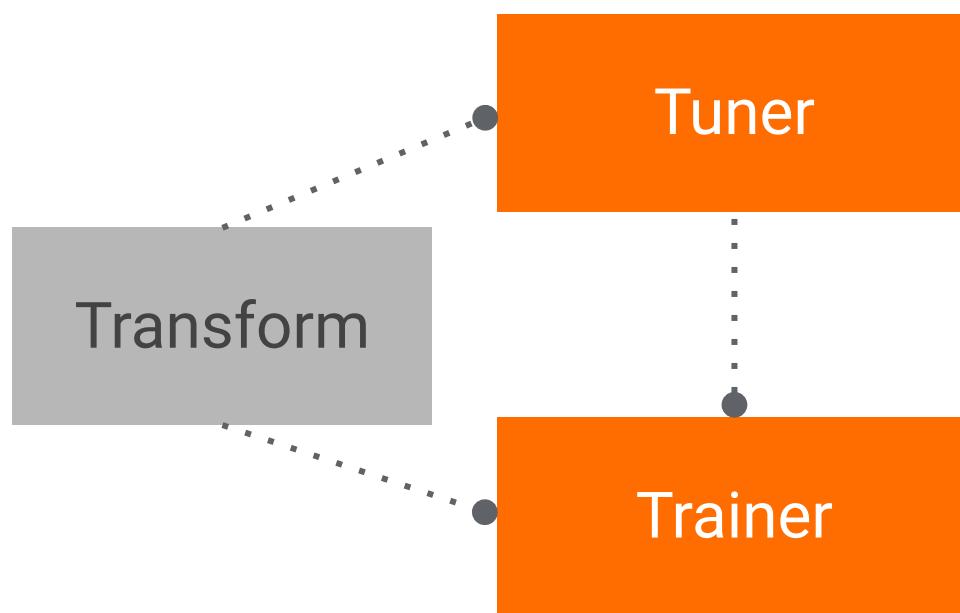
```
trainer = Trainer(  
    module_file=module_file, # Contains `run_fn`.  
    custom_executor_spec=executor_spec.  
    ExecutorClassSpec(GenericExecutor),  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    schema=schema_gen.outputs['schema'],  
    # This will be passed to `run_fn`.  
    hyperparameters=tuner.outputs['best_hyperparameters'],  
    train_args=trainer_pb2.TrainArgs(num_steps=100),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

## ML project lifecycle benefits

- Supports KerasTuner library for hyperparameter tuning and integration with TensorFlow Trainer

# Component: Tuner

## Tuner works with Trainer



## Configuration

```
tuner = Tuner(  
    module_file=module_file, # Contains `tuner_fn`.  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=20),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

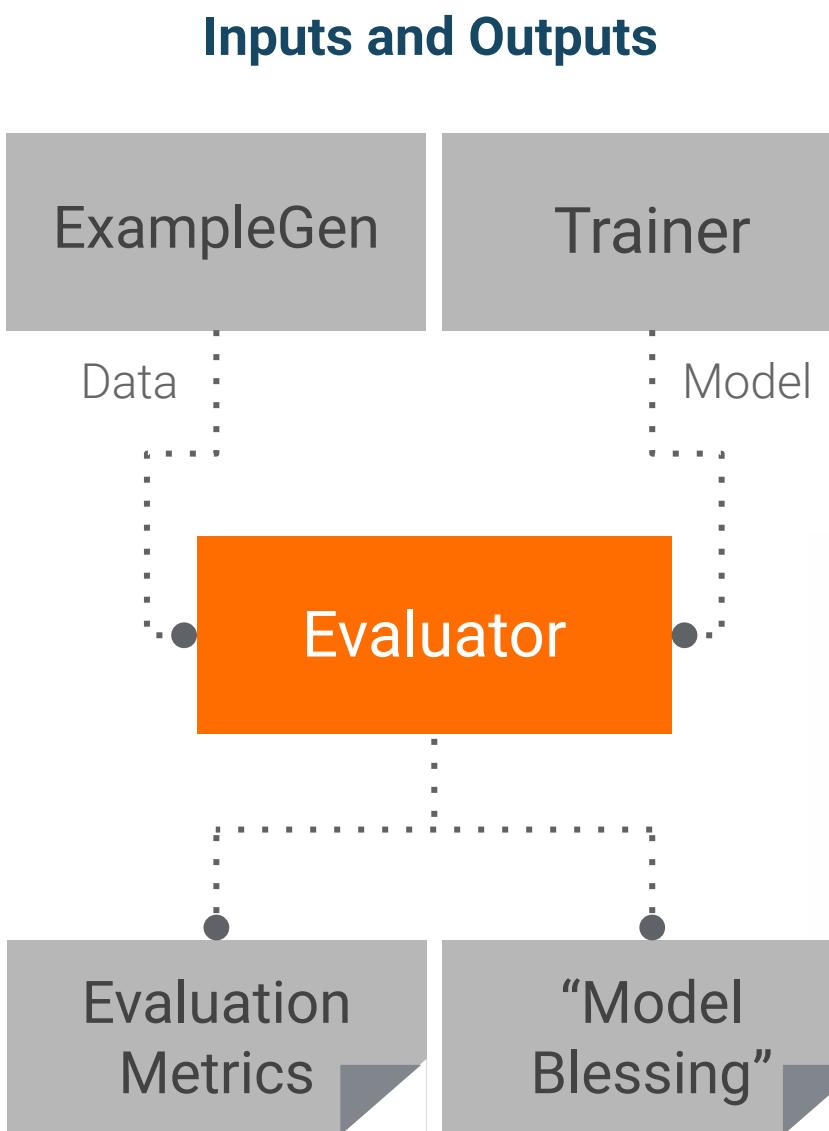
## Best hyperparameters added to Trainer in pipeline

```
trainer = Trainer(  
    module_file=module_file, # Contains `run_fn`.  
    custom_executor_spec=executor_spec,  
    ExecutorClassSpec(GenericExecutor),  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    schema=schema_gen.outputs['schema'],  
    # This will be passed to `run_fn`.  
    hyperparameters=tuner.outputs['best_hyperparameters'],  
    train_args=trainer_pb2.TrainArgs(num_steps=100),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

## ML project lifecycle benefits

- Supports KerasTuner library for hyperparameter tuning and integration with TensorFlow Trainer
- Can use Google Cloud AI Platform Optimizer for distributed tuning

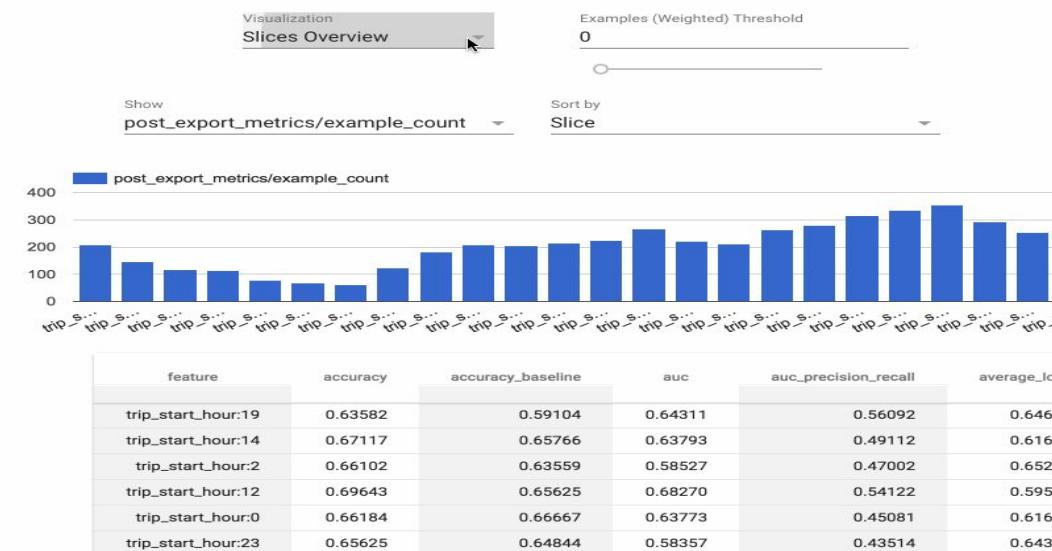
# Component: Evaluator



## Configuration

```
model_analyzer = Evaluator(  
    examples=example_gen.outputs.examples,  
    model=trainer.outputs.model,  
    baseline_model=model_resolver.outputs.model,  
    eval_config=eval_config)
```

## TensorFlow Model Analysis (TFMA) library for visualizing model evaluation



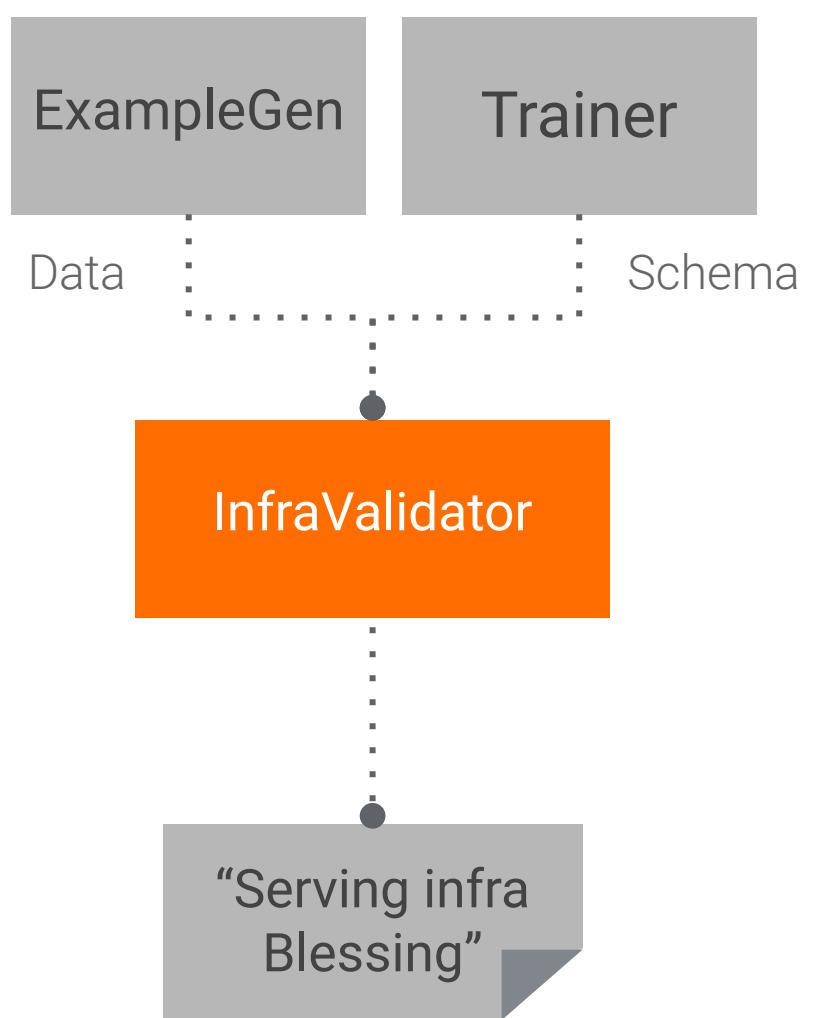
## ML project lifecycle benefits

- Uses TensorFlow Model Analysis (TFMA) and Apache Beam for scalable model evaluation across data splits and slices.
- Validate model performance “good enough” to be pushed to production

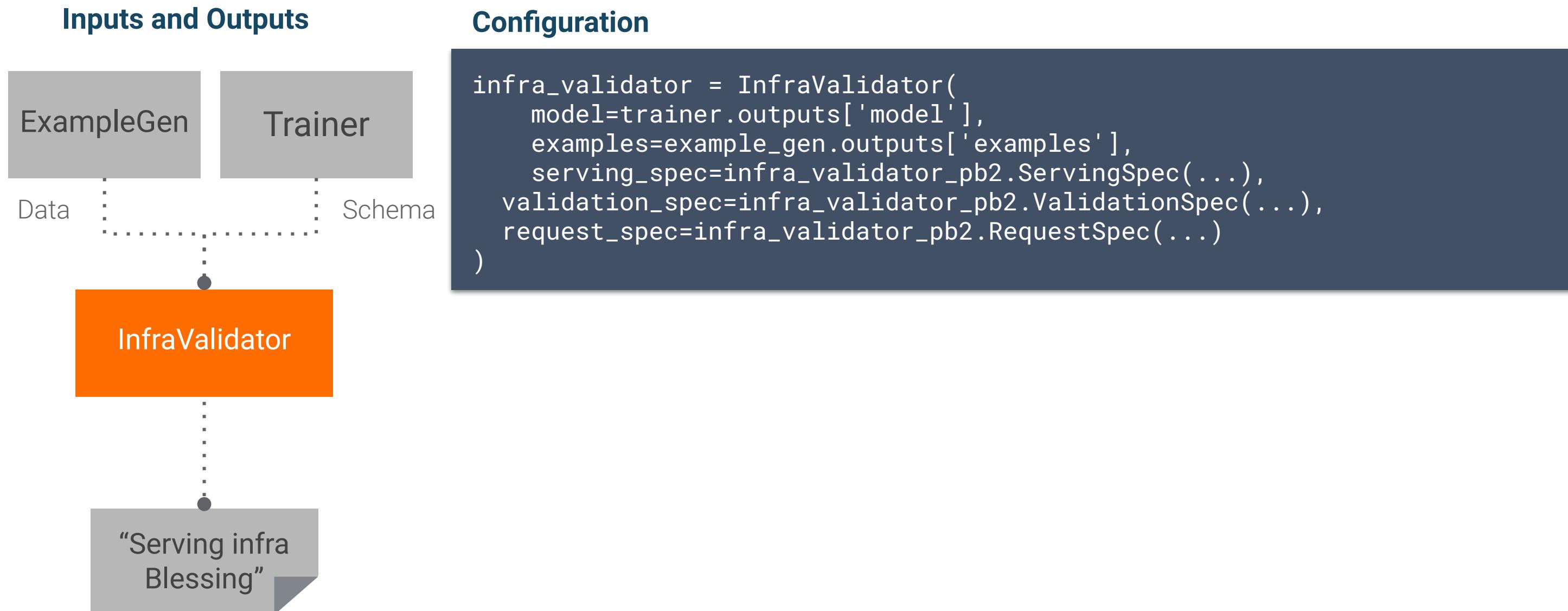


# Component: InfraValidator

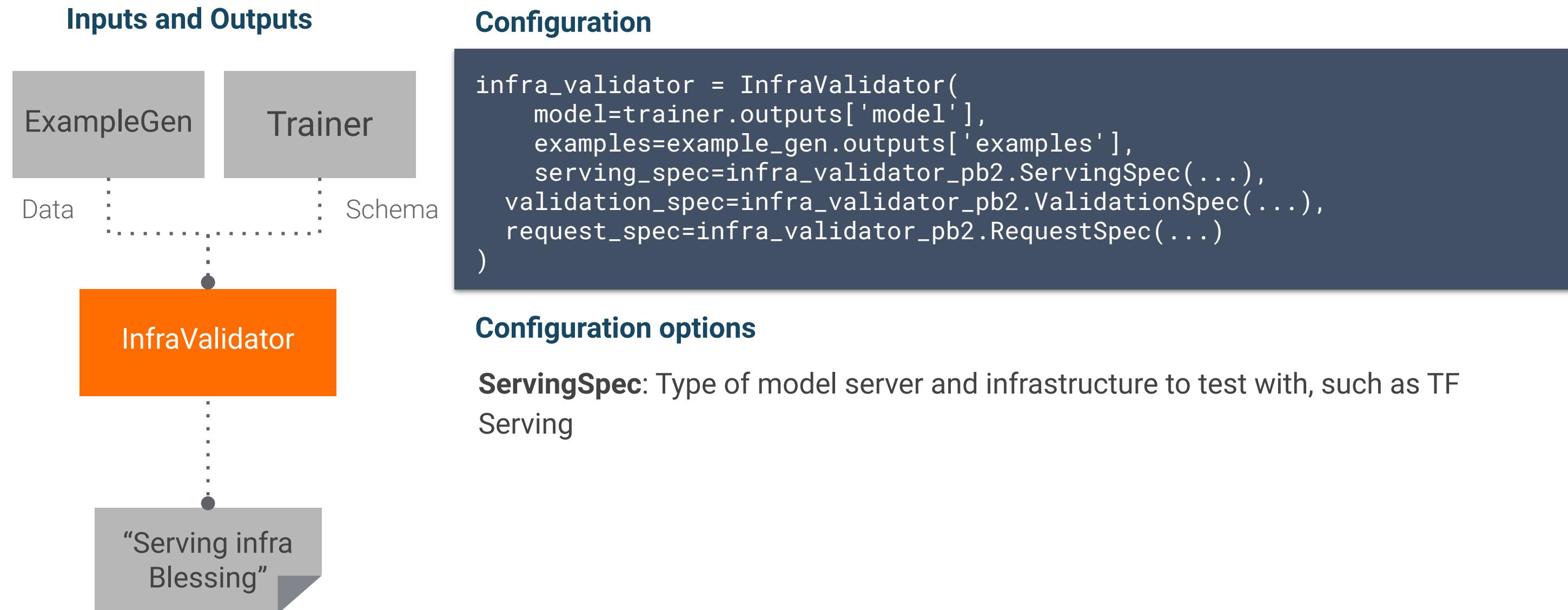
## Inputs and Outputs



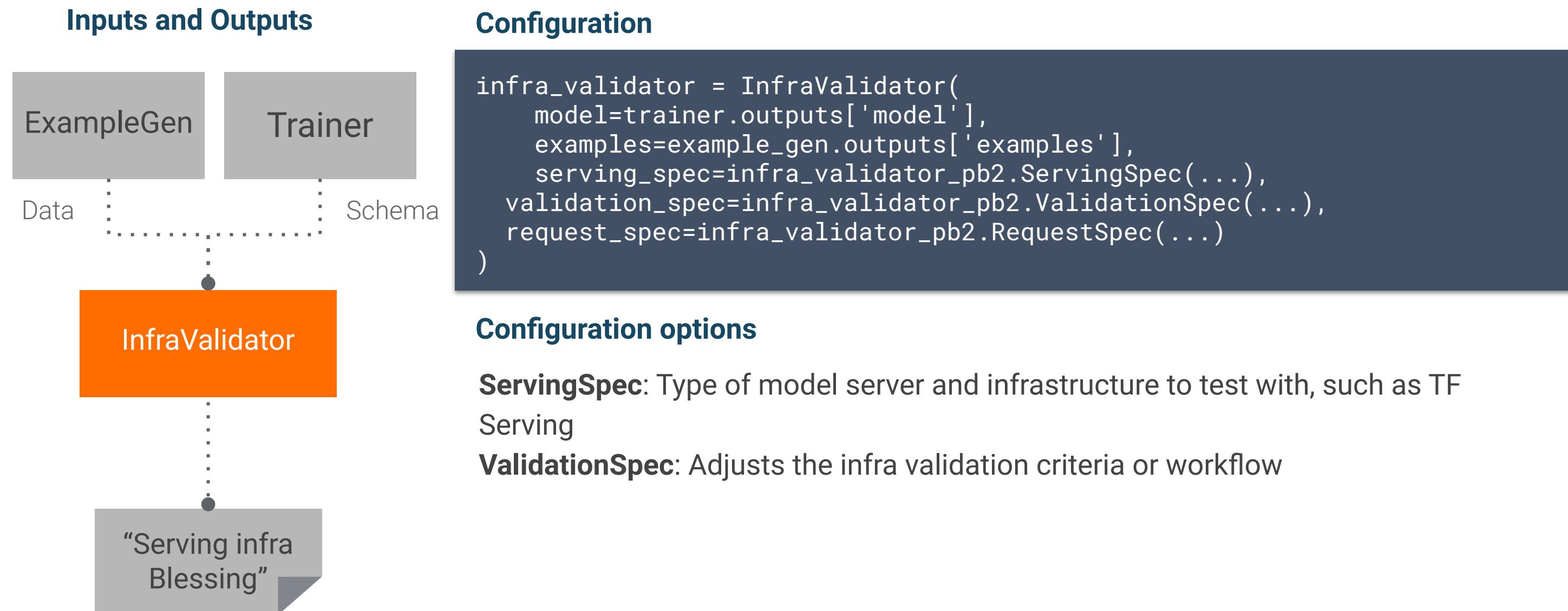
# Component: InfraValidator



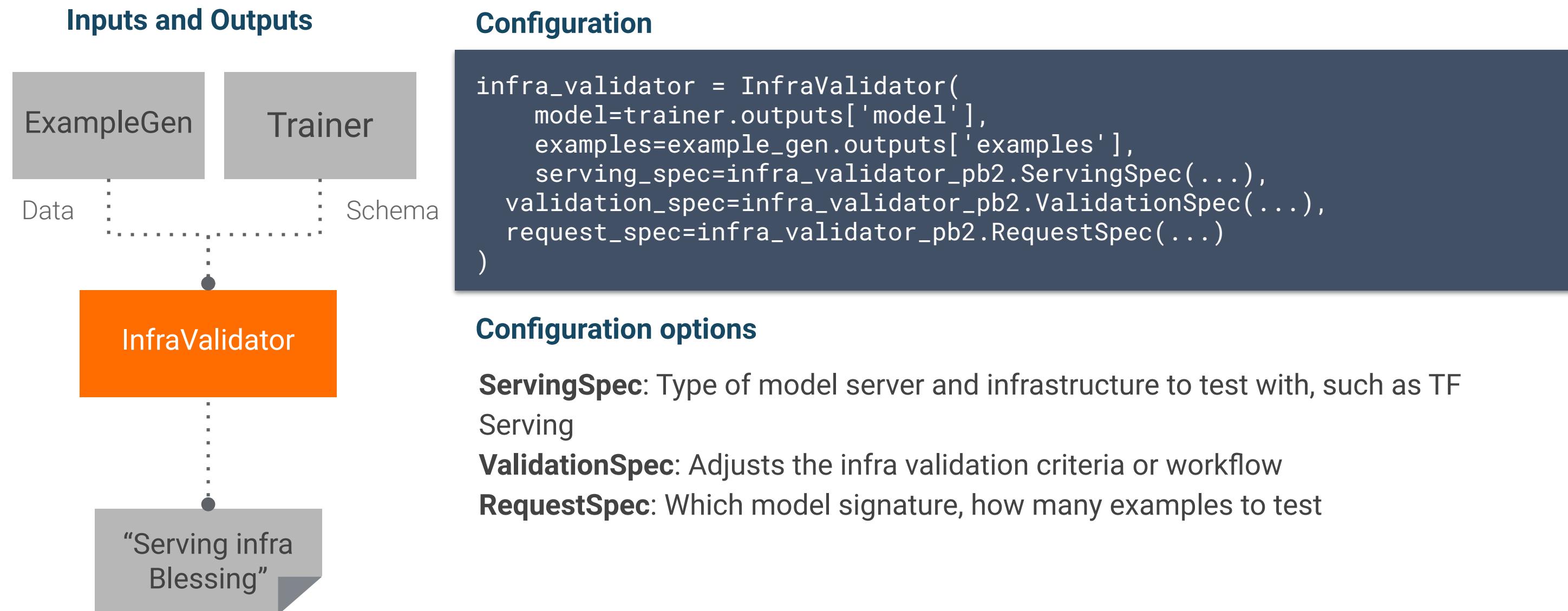
# Component: InfraValidator



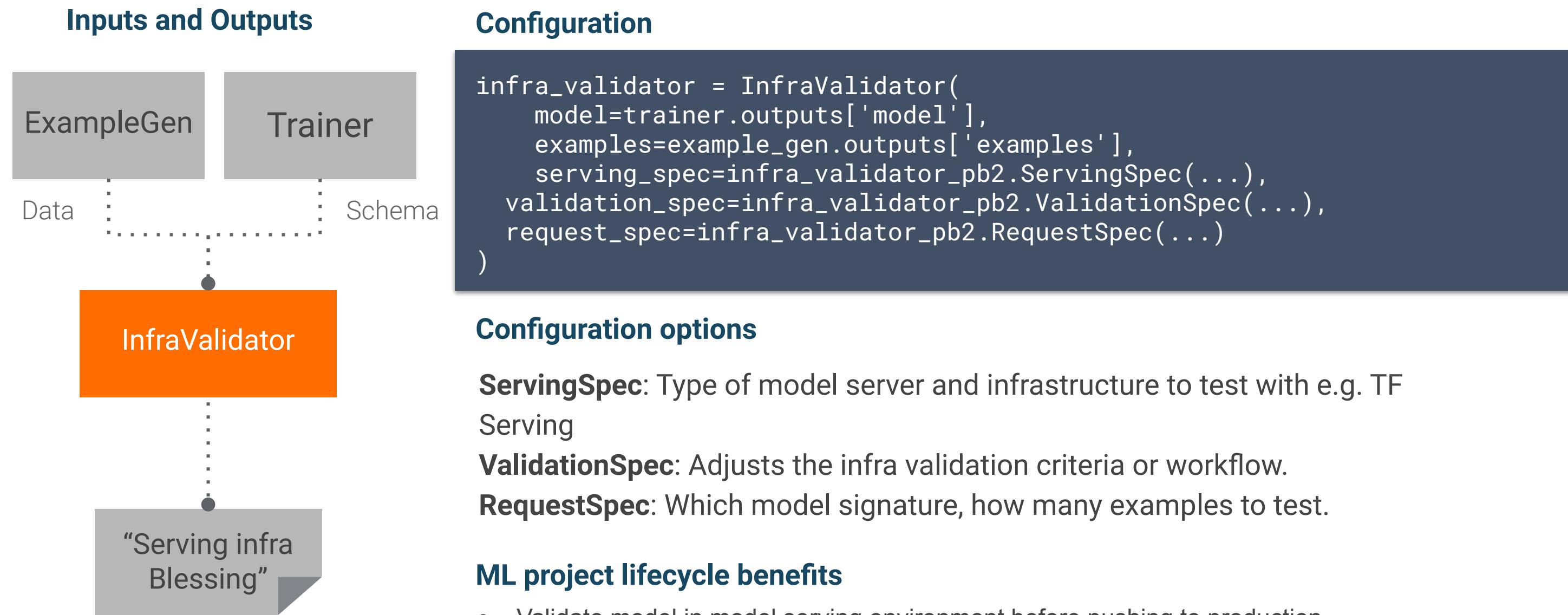
# Component: InfraValidator



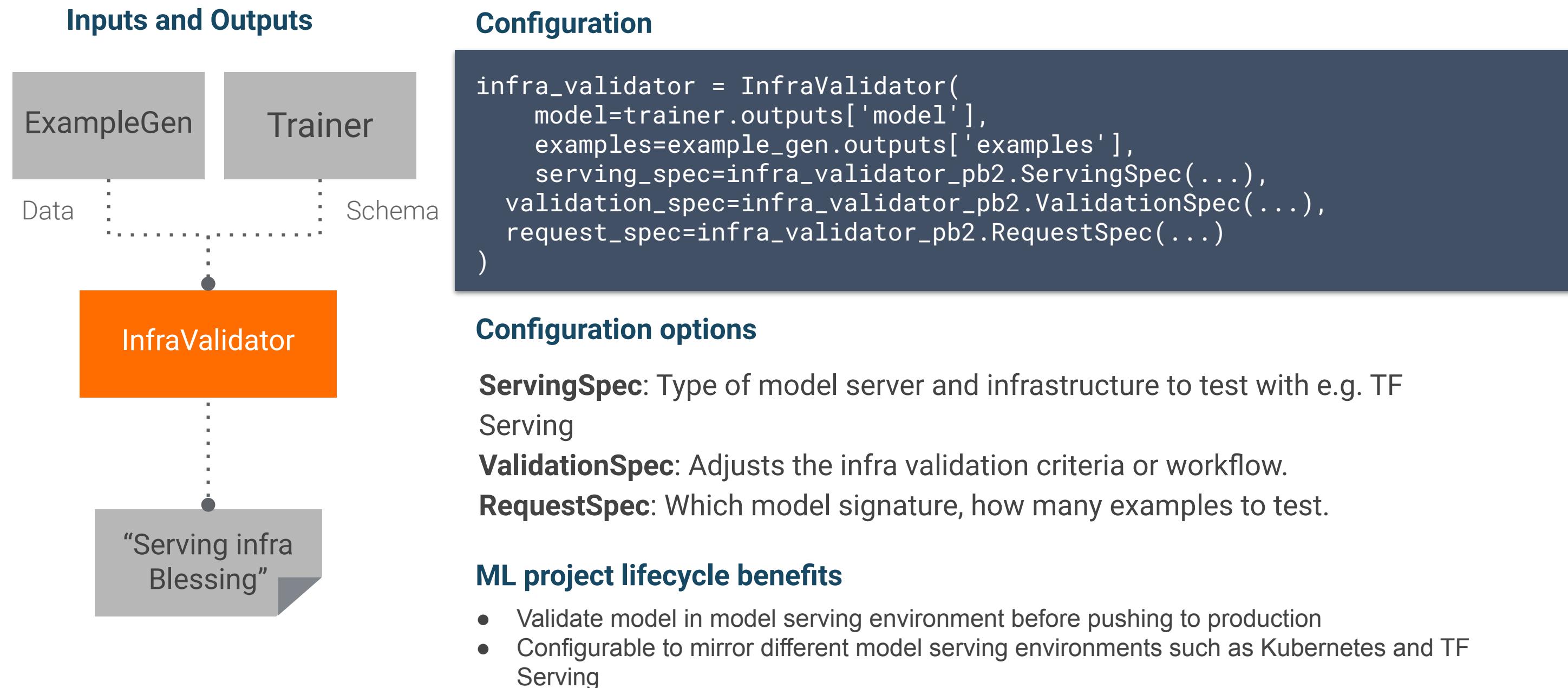
# Component: InfraValidator



# Component: InfraValidator

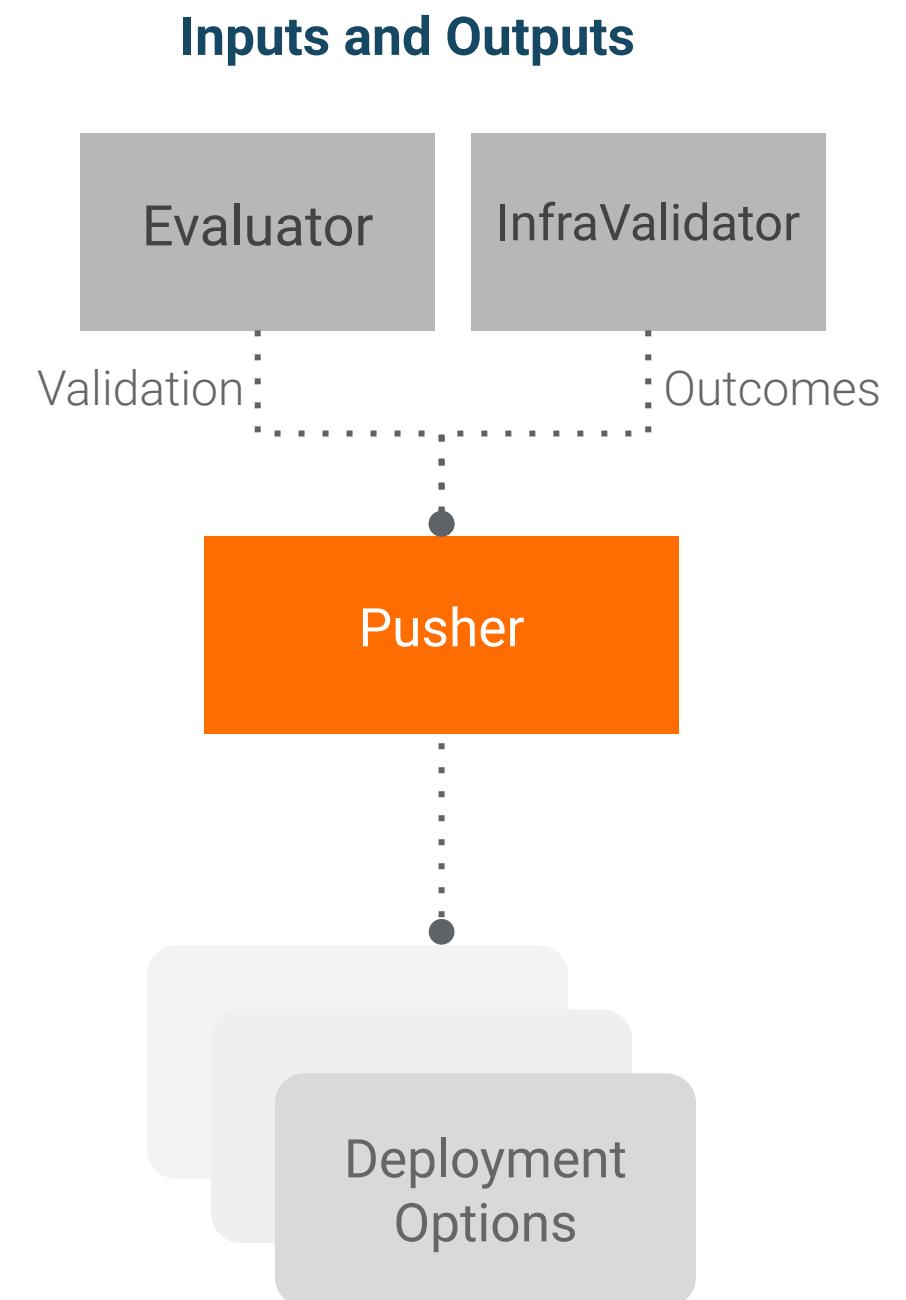


# Component: InfraValidator

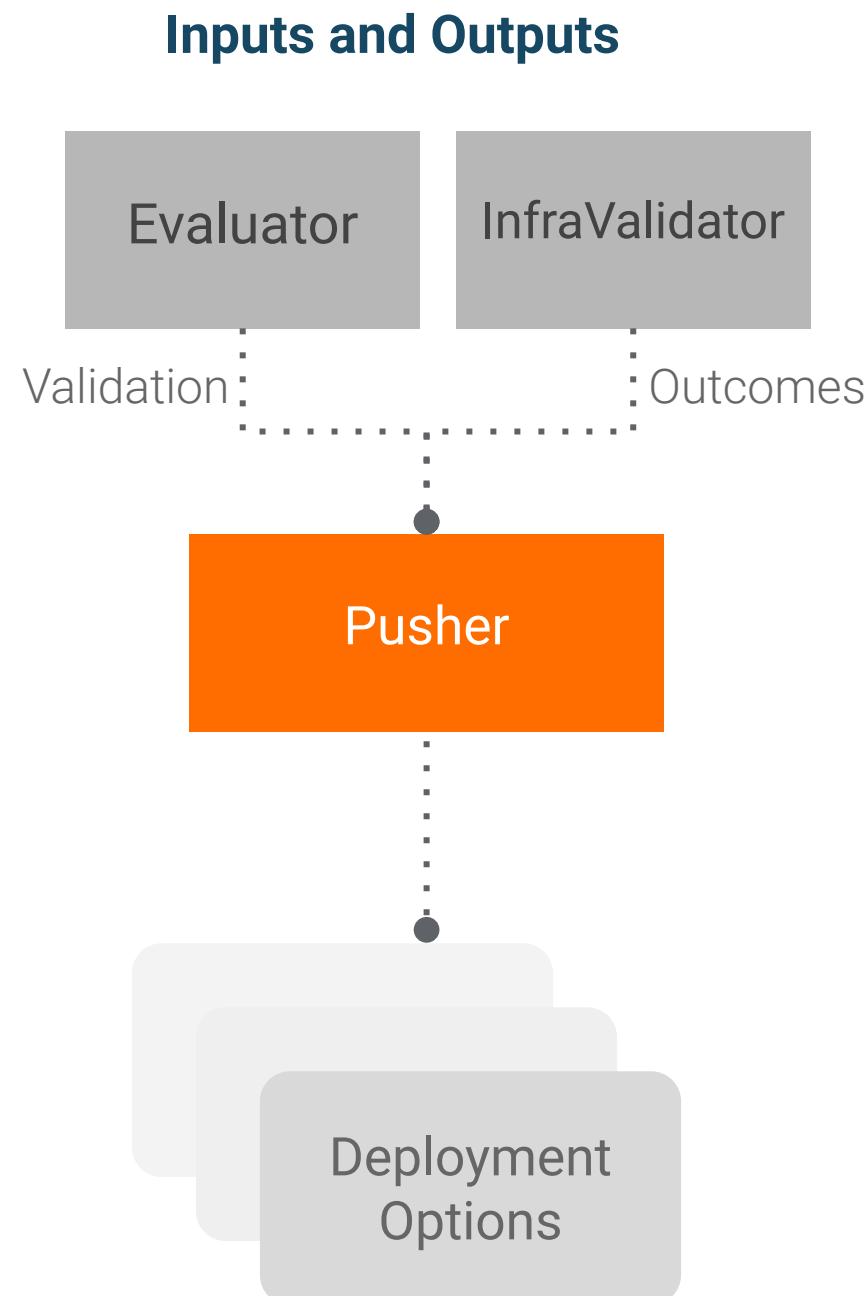


---

# Component: Pusher



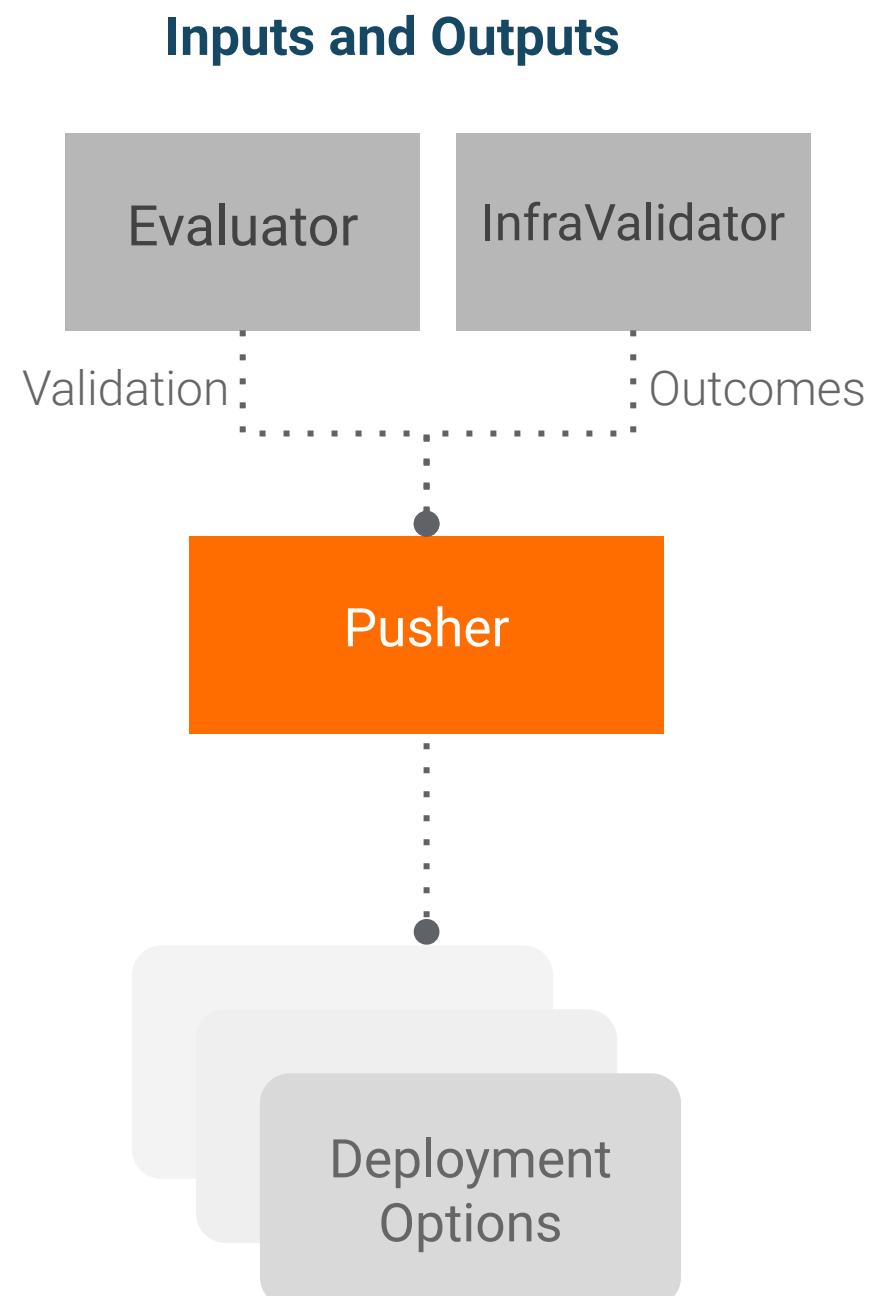
# Component: Pusher



## Configuration

```
pusher = Pusher(  
    model=trainer.outputs['model'],  
    model_blessing=model_analyzer.outputs['blessing'],  
    infra_blessing=infra_validator.outputs['blessing'],  
    push_destination=pusher_pb2.PushDestination(  
        filesystem=pusher_pb2.PushDestination.Filesystem(  
            base_directory=SERVING_MODEL_DIR)))
```

# Component: Pusher



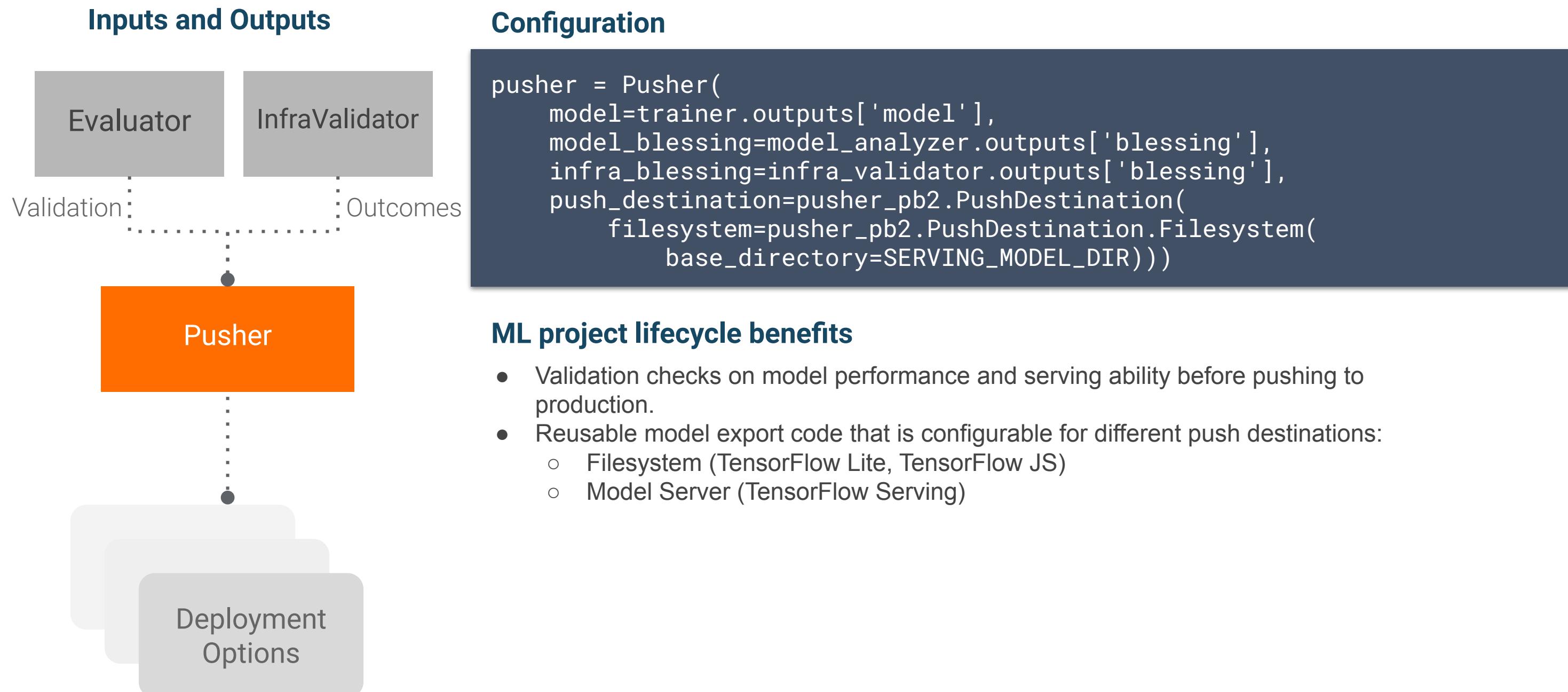
## Configuration

```
pusher = Pusher(  
    model=trainer.outputs['model'],  
    model_blessing=model_analyzer.outputs['blessing'],  
    infra_blessing=infra_validator.outputs['blessing'],  
    push_destination=pusher_pb2.PushDestination(  
        filesystem=pusher_pb2.PushDestination.Filesystem(  
            base_directory=SERVING_MODEL_DIR)))
```

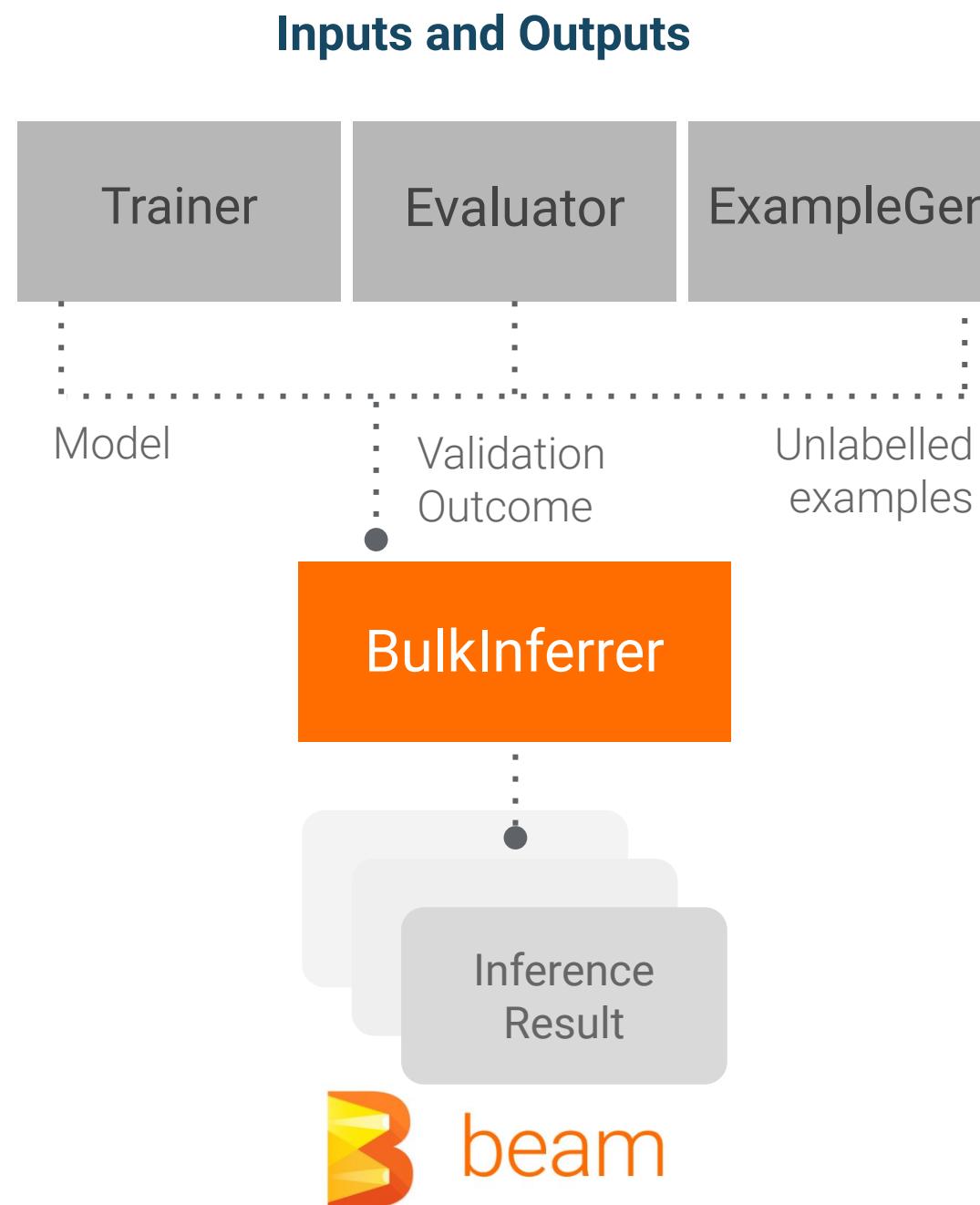
## ML project lifecycle benefits

- Validation checks on model performance and serving ability before pushing to production.

# Component: Pusher



# Component: BulkInferer



## Configuration

```
bulk_inferrer = BulkInferer(  
    examples=inference_example_gen.outputs['examples'],  
    model_export=trainer.outputs['output'],  
    model_blessing=evaluator.outputs['blessing'],  
    data_spec=bulk_inferrer_pb2.DataSpec(  
        example_splits=['unlabelled']),  
    model_spec=bulk_inferrer_pb2.ModelSpec())
```

## Configuration options

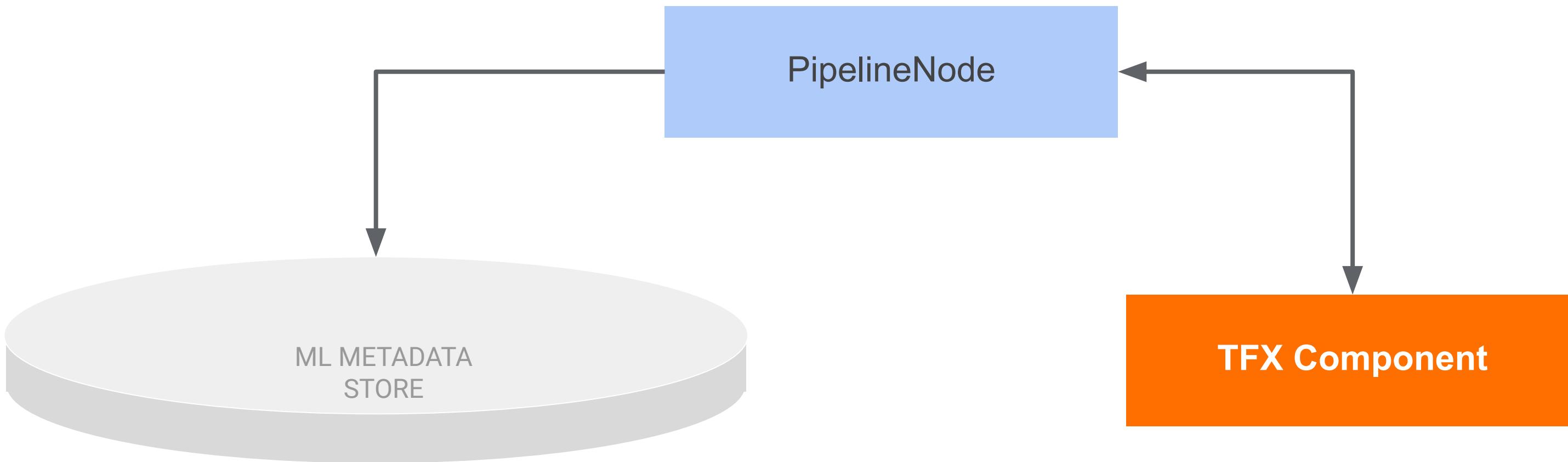
- Block batch inference on a successful model validation.
- Choose the inference examples from example gen's output.
- Choose the signatures and tags of inference model.

## ML project lifecycle benefits

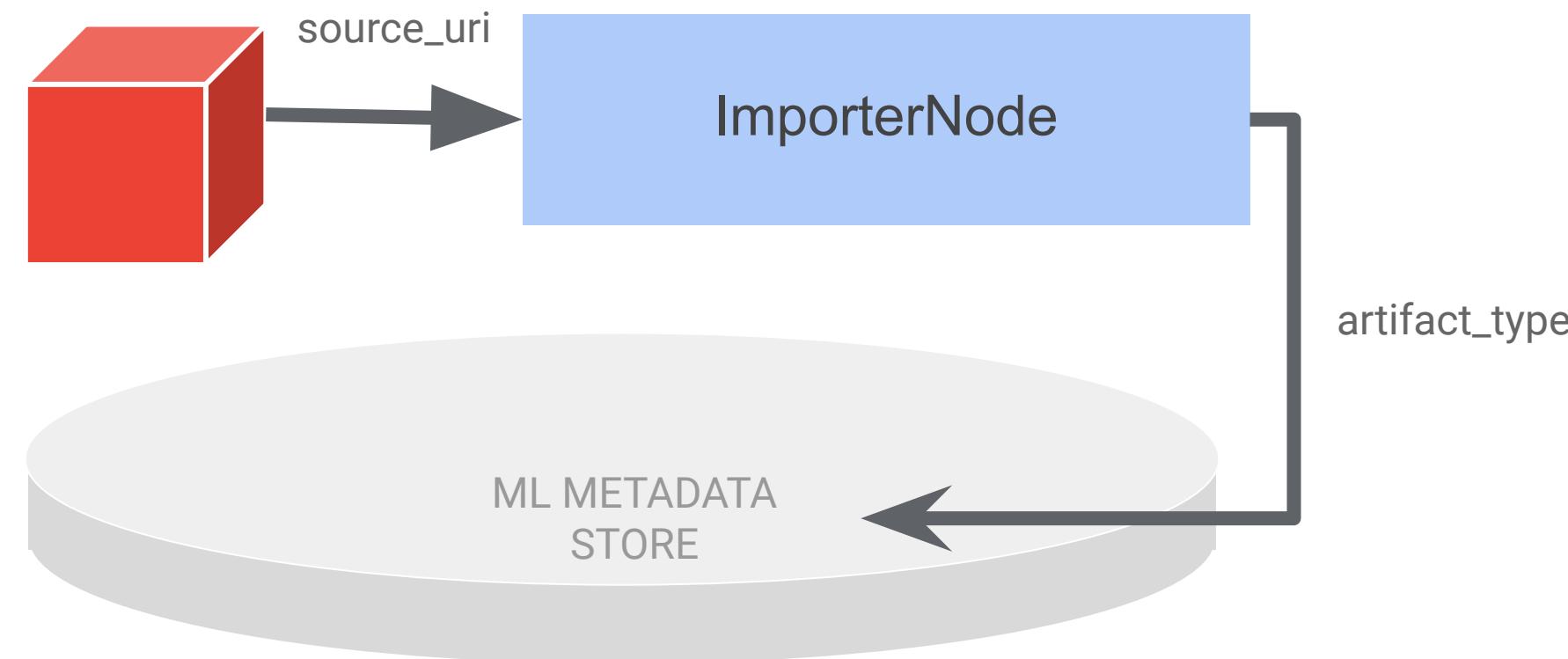
- Task and data-driven inference directly in your TFX pipeline

---

# What are pipeline nodes?

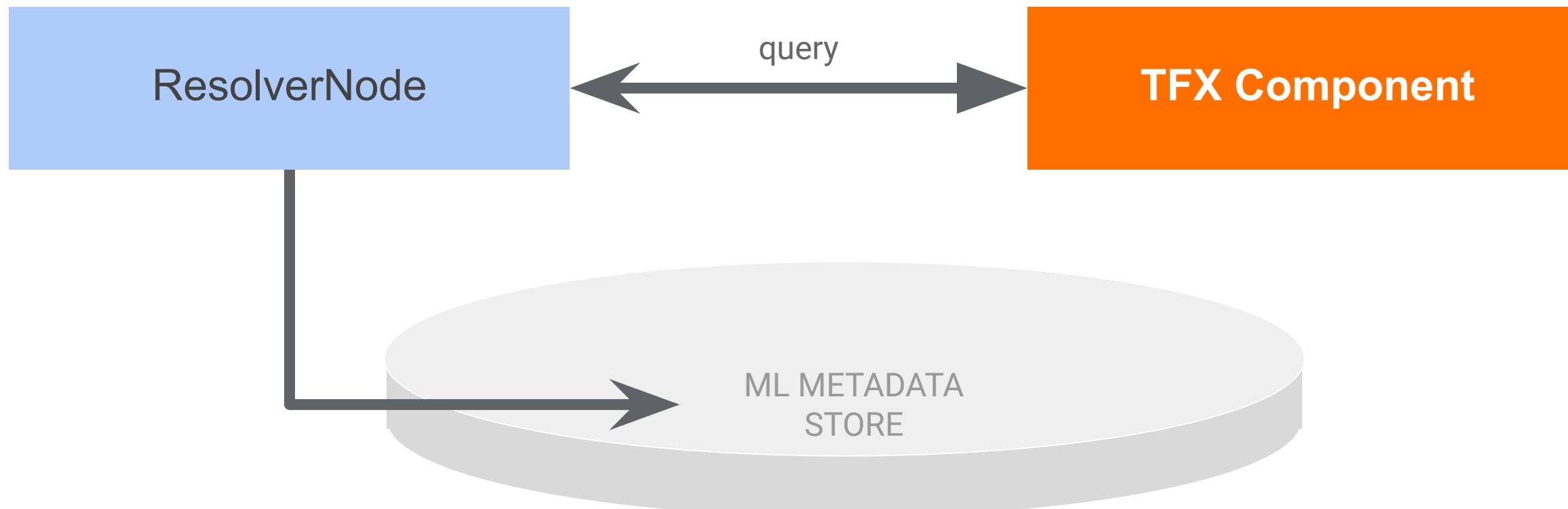


# ImporterNode imports an external data object into ML Metadata



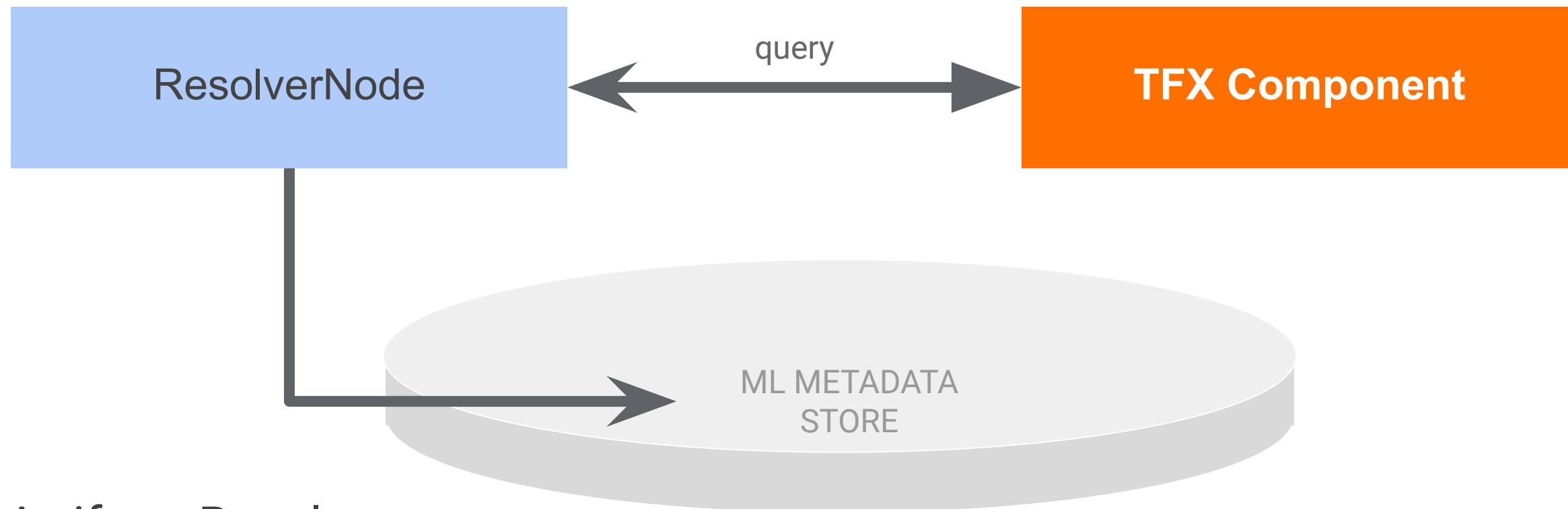
```
importer = ImporterNode(  
    instance_name='import_schema',  
    source_uri='uri/to/schema'  
    artifact_type=standard_artifacts.Schema,  
    reimport=False)
```

# ResolverNode performs metadata queries



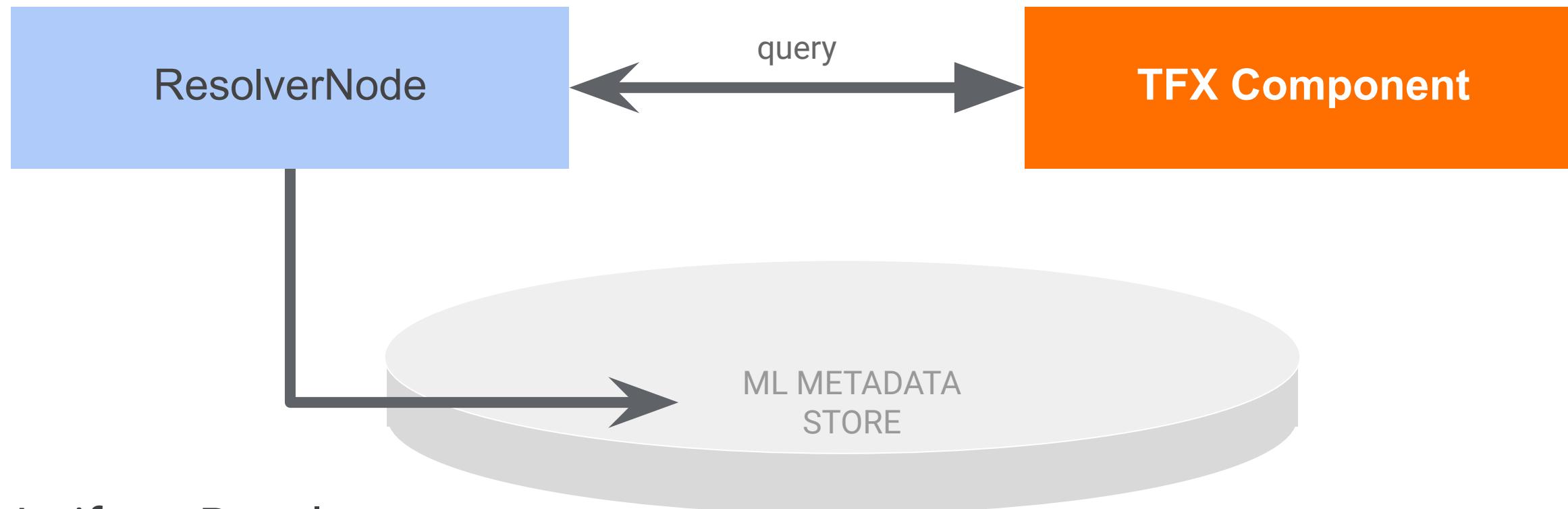
```
latest_five_examples_resolver = ResolverNode(  
    instance_name='latest_five_examples_resolver',  
    resolver_class=latest_artifacts_resolver.LatestArtifactsResolver, # Class  
    resolver_config={'desired_num_of_artifacts' : 5}, # Query  
    config  
    examples=example_gen.outputs['examples']) # Channel
```

# Current ResolverNodes



- LatestArtifactsResolver

# Current ResolverNodes



- LatestArtifactsResolver
- LatestBlessedModelResolver

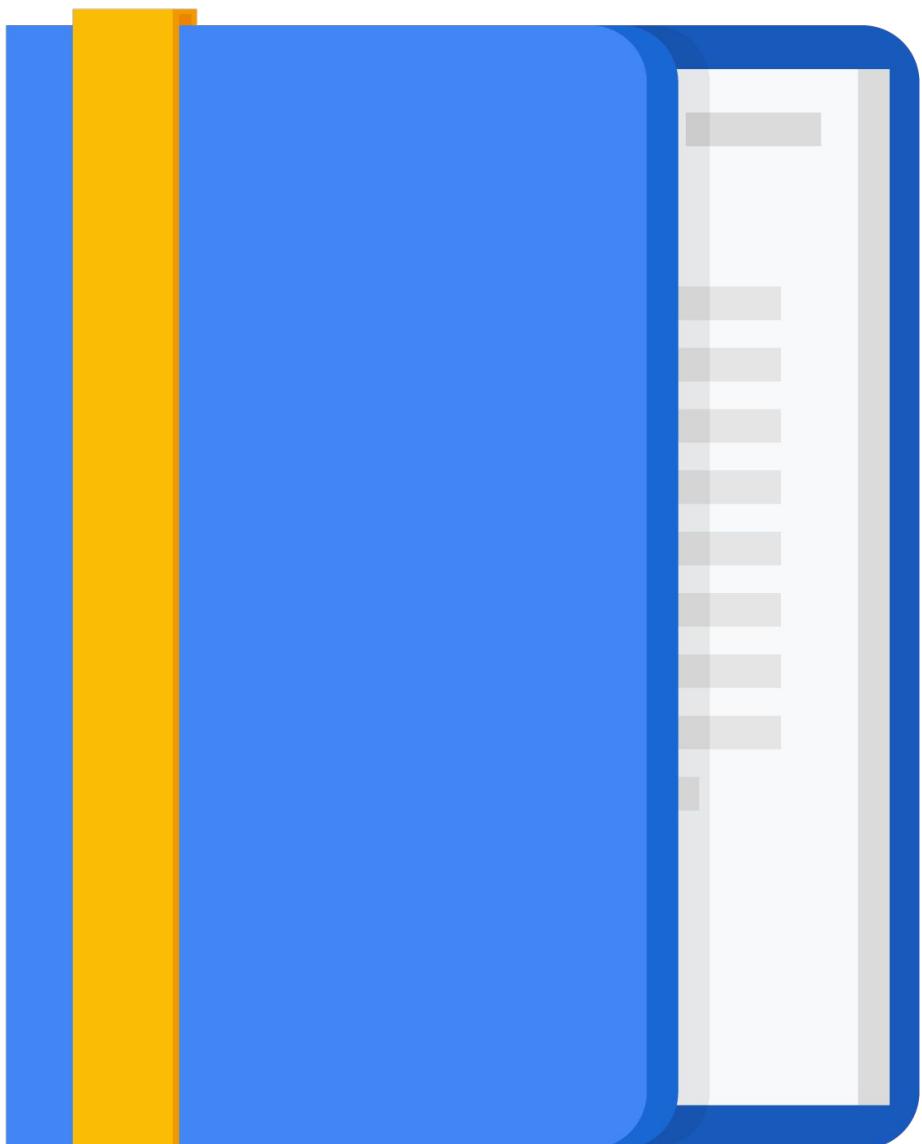
---

# Agenda

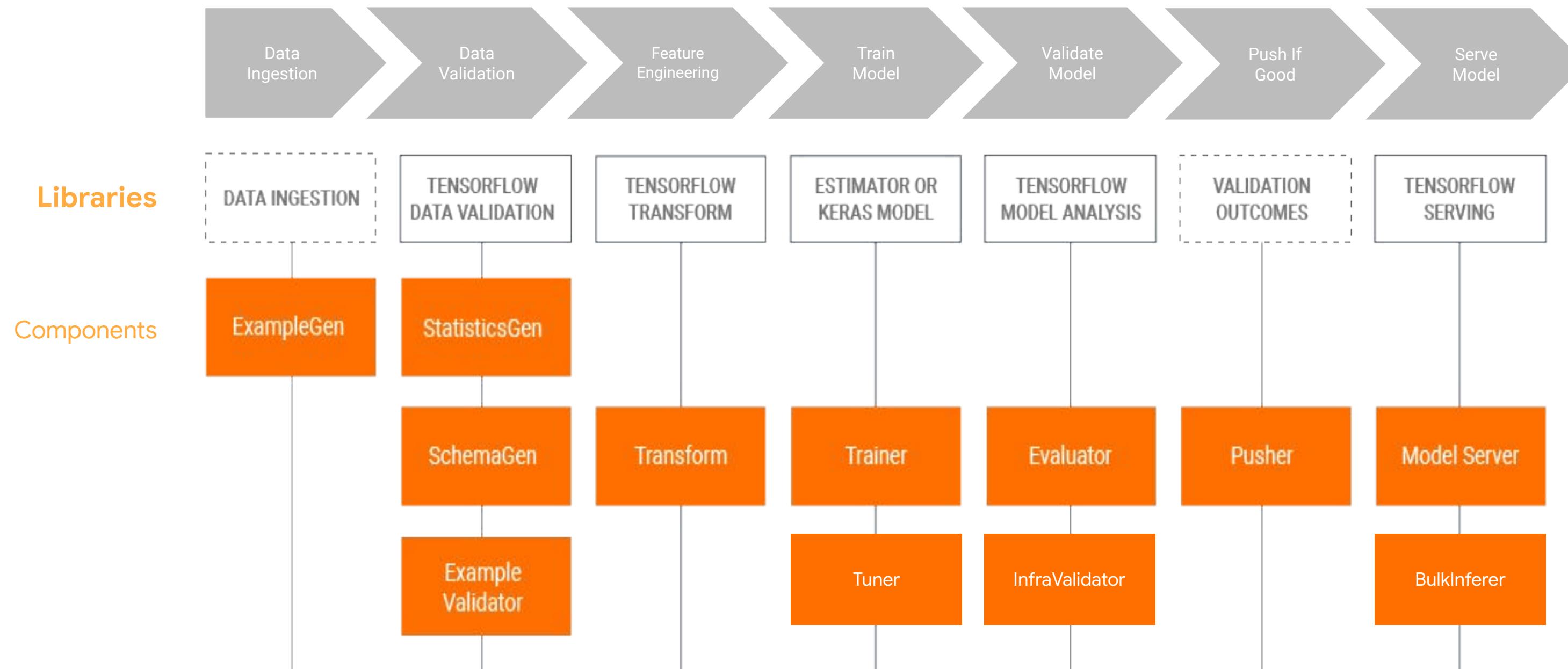
TensorFlow Extended (TFX)

TFX standard components

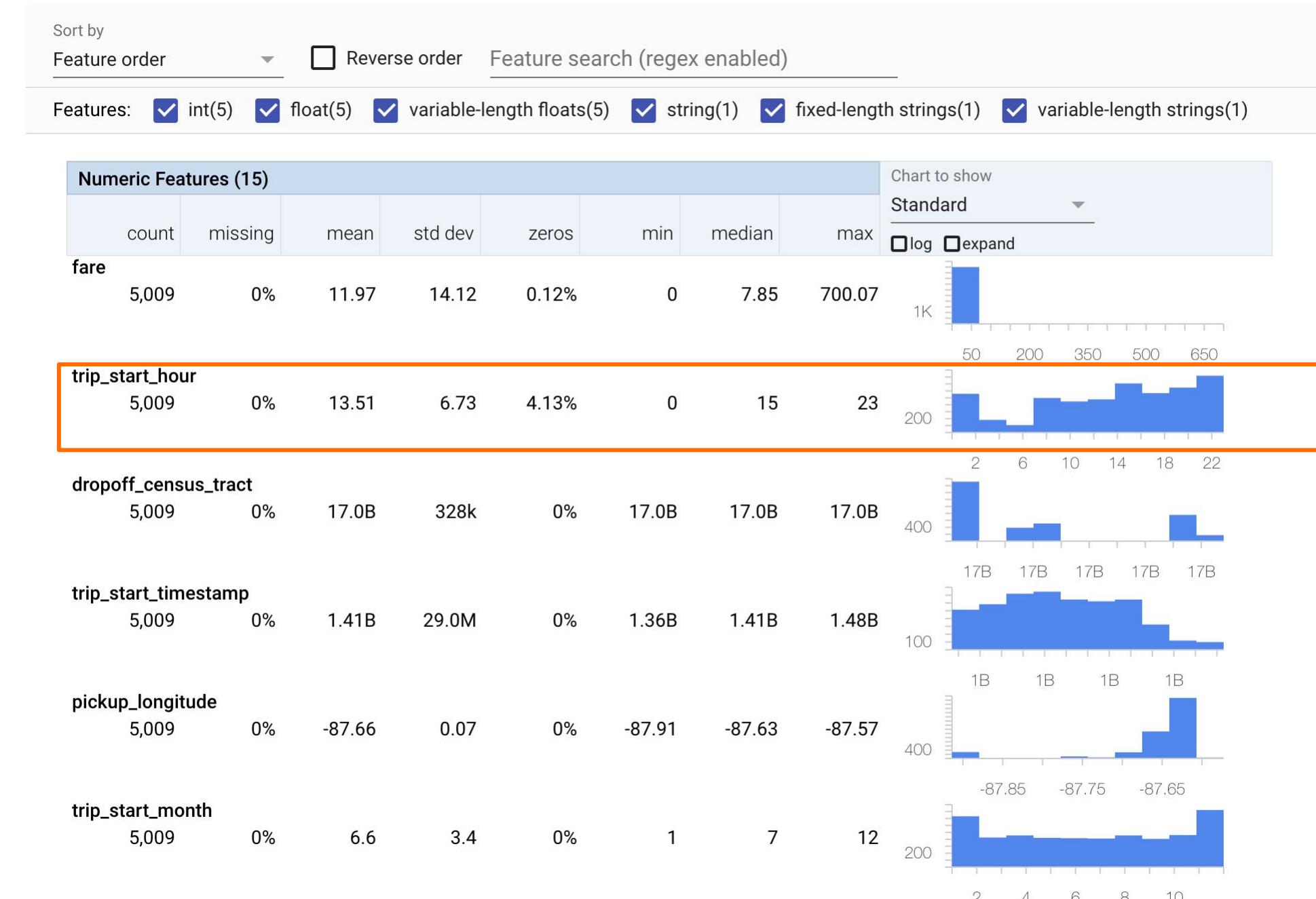
**TFX libraries**



# TFX libraries integrate with each phase of a machine learning pipeline



# TensorFlow Data Validation (TFDV) for analyzing and validating data



---

# TensorFlow Transform (TFT) library for preprocessing data and feature engineering with TensorFlow

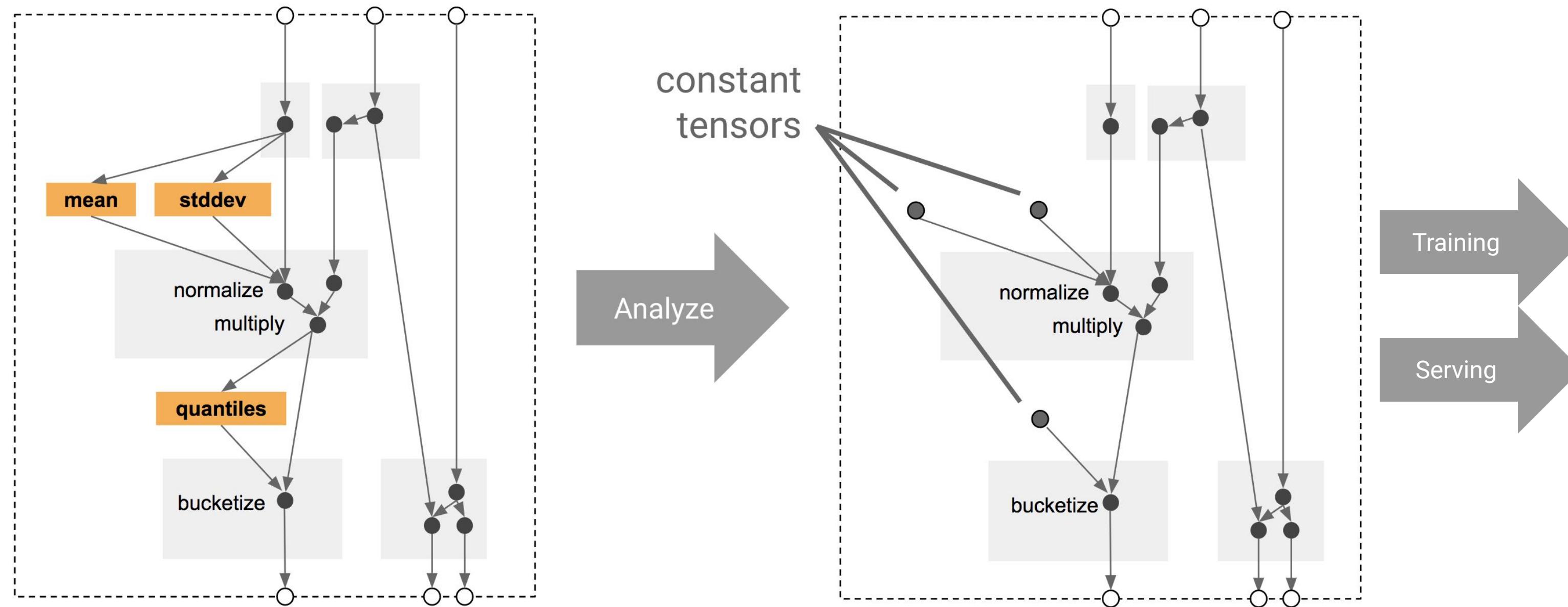
## Analyzers

- max
- min
- mean
- sum
- var
- covariance
- quantiles
- size
- vocabulary
- pca

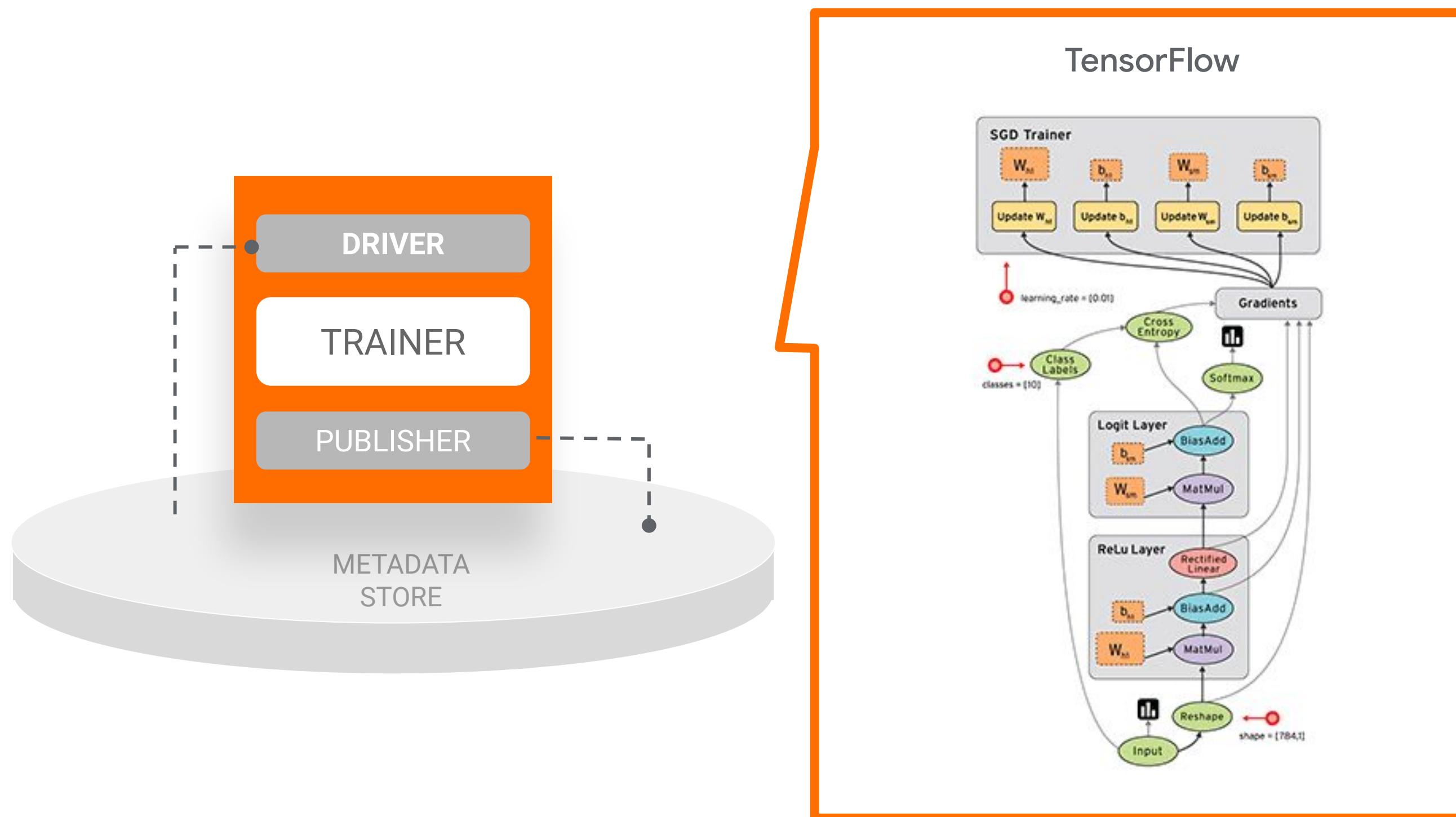
## Mappers

- bucketize
- apply\_buckets
- hash\_string
- ngram
- scale\_0+1
- scale\_z\_score
- Scale\_max\_min
- tfidf
- compute\_and\_apply\_vocabulary

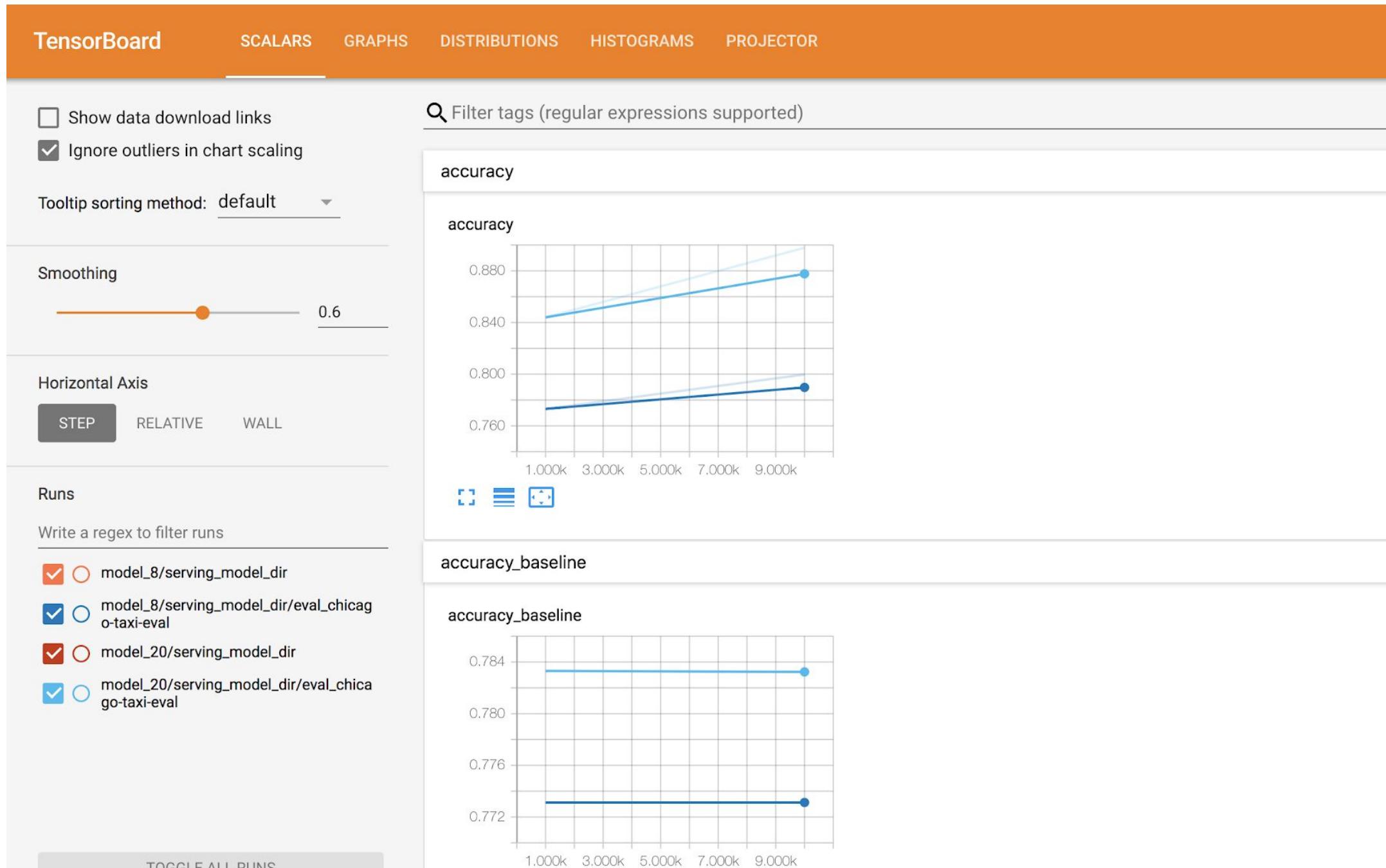
# TensorFlow Transform (TFT) library for preprocessing data and feature engin with TensorFlow



# TensorFlow for model training and tuning



# TensorBoard visualization and tooling for ML experimentation integrated with TFX

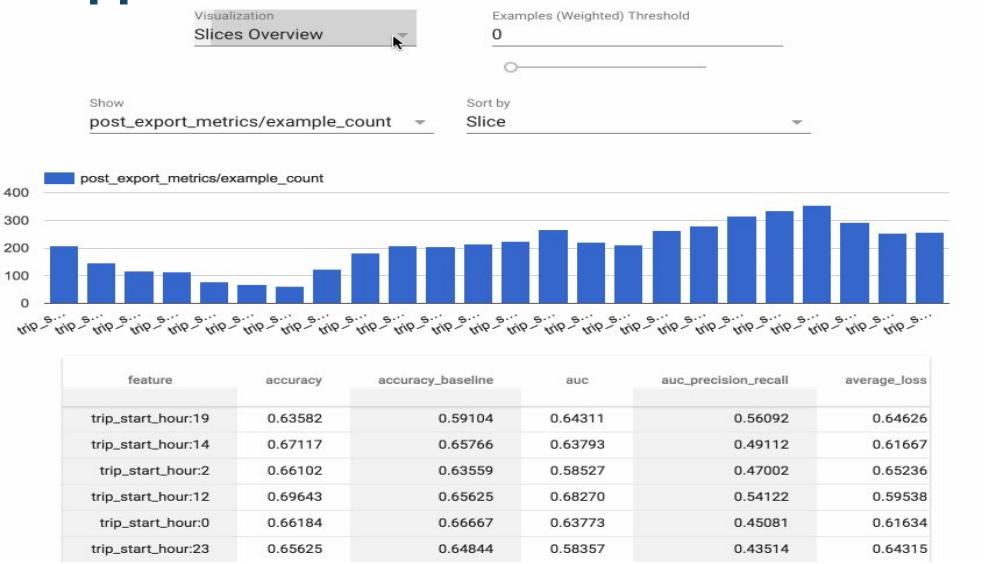


---

# **TensorFlow Model Analysis (TFMA) library for model evaluation**

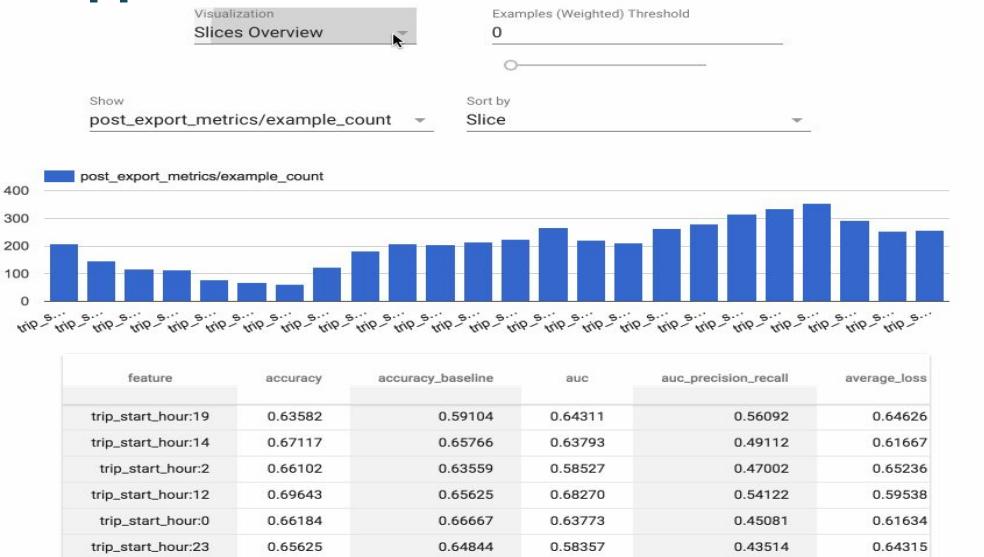
# TensorFlow Model Analysis (TFMA) library for model evaluation

TFMA supports interactive model evaluation

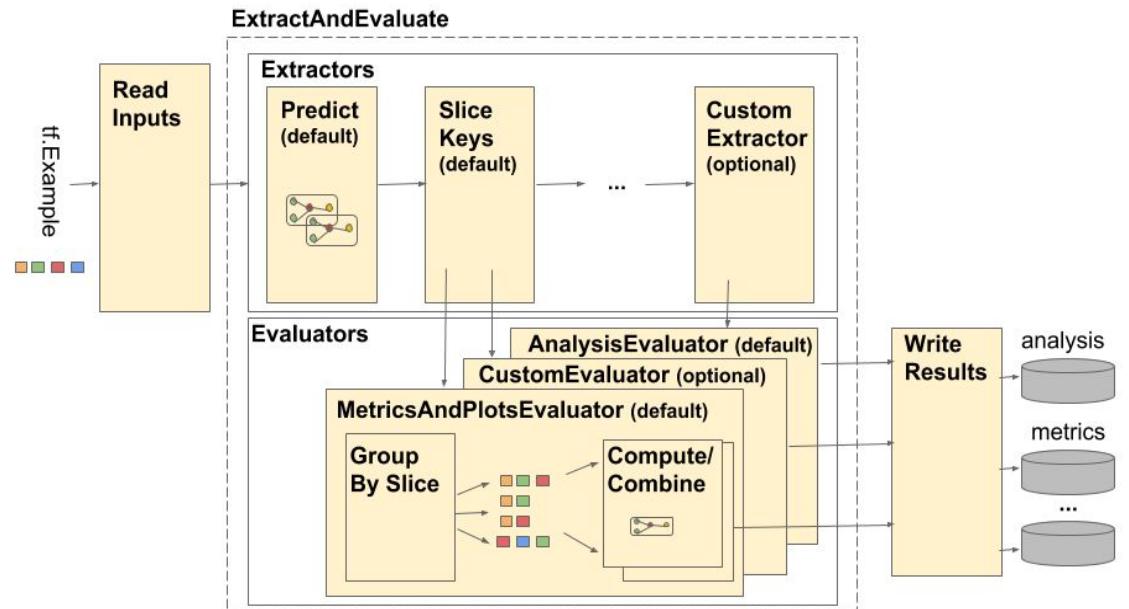


# TensorFlow Model Analysis (TFMA) library for model evaluation

TFMA supports interactive model evaluation

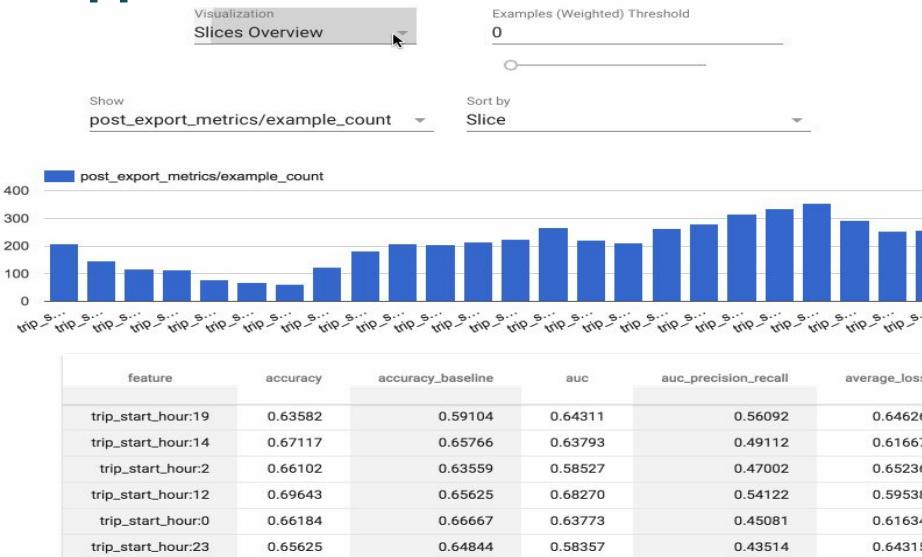


TFMA scales model evaluation with Apache Beam

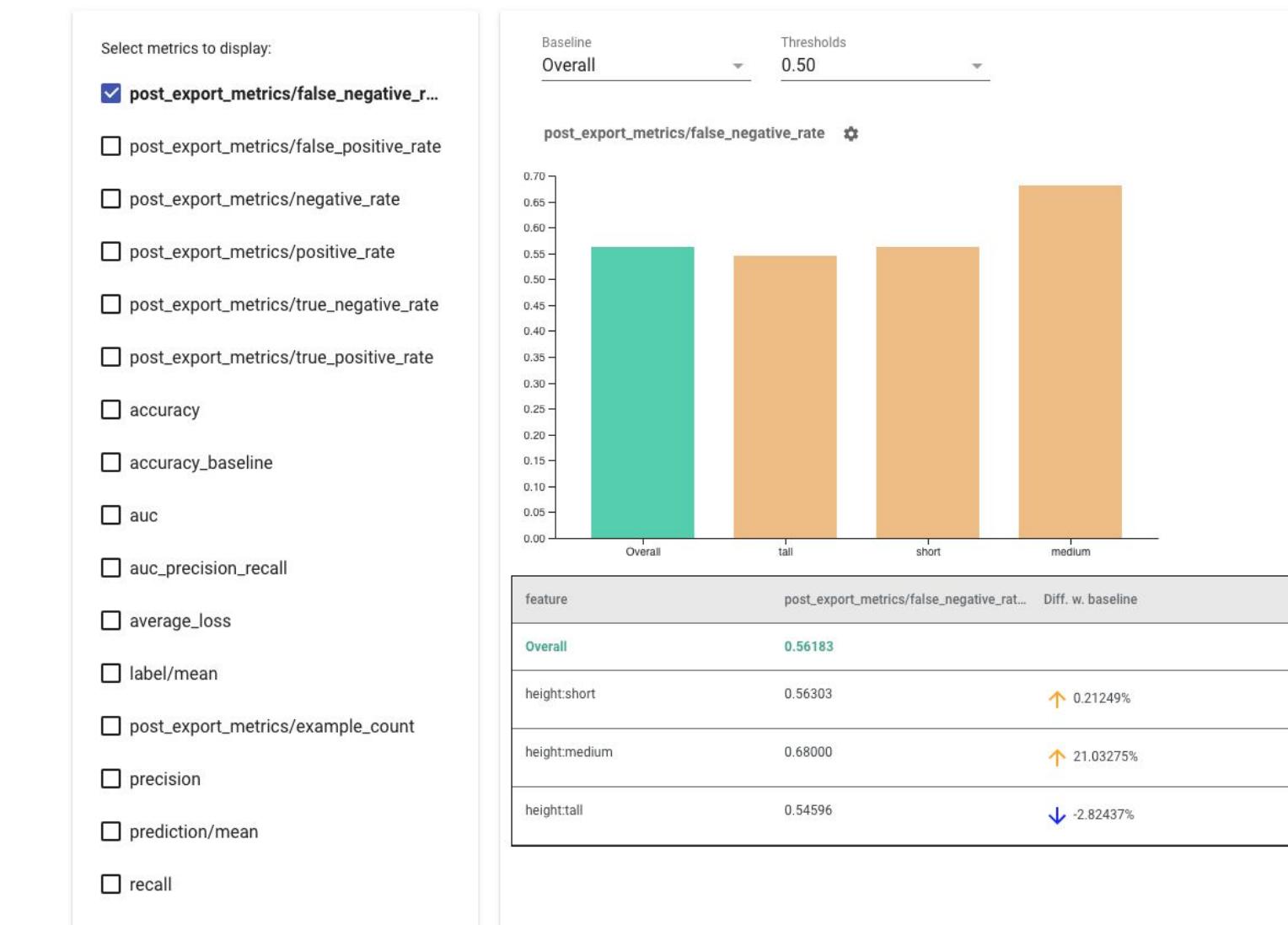


# TensorFlow Model Analysis (TFMA) library for model evaluation

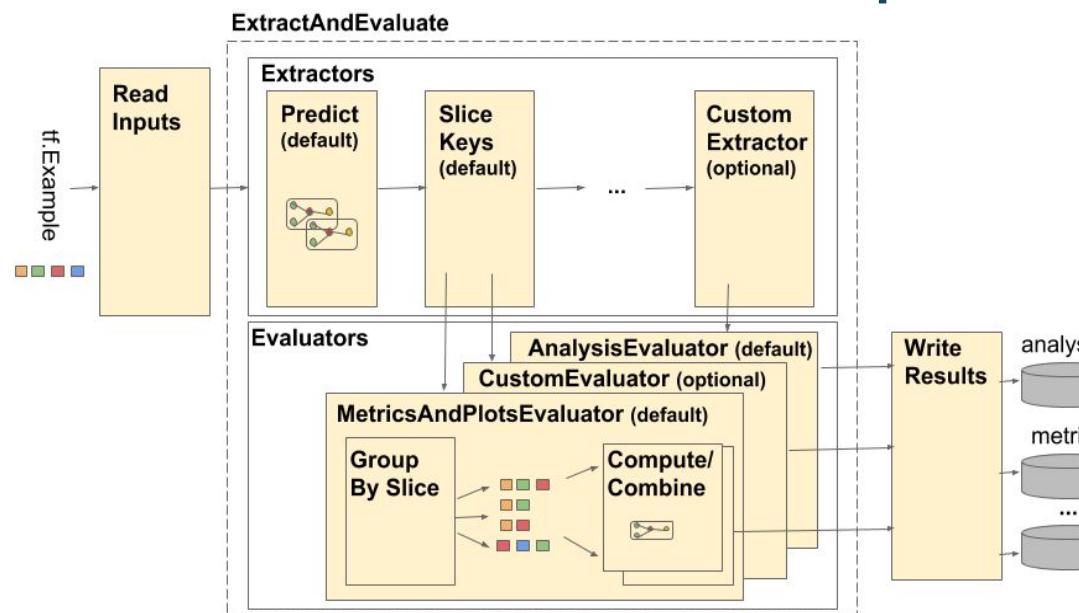
## TFMA supports interactive model evaluation



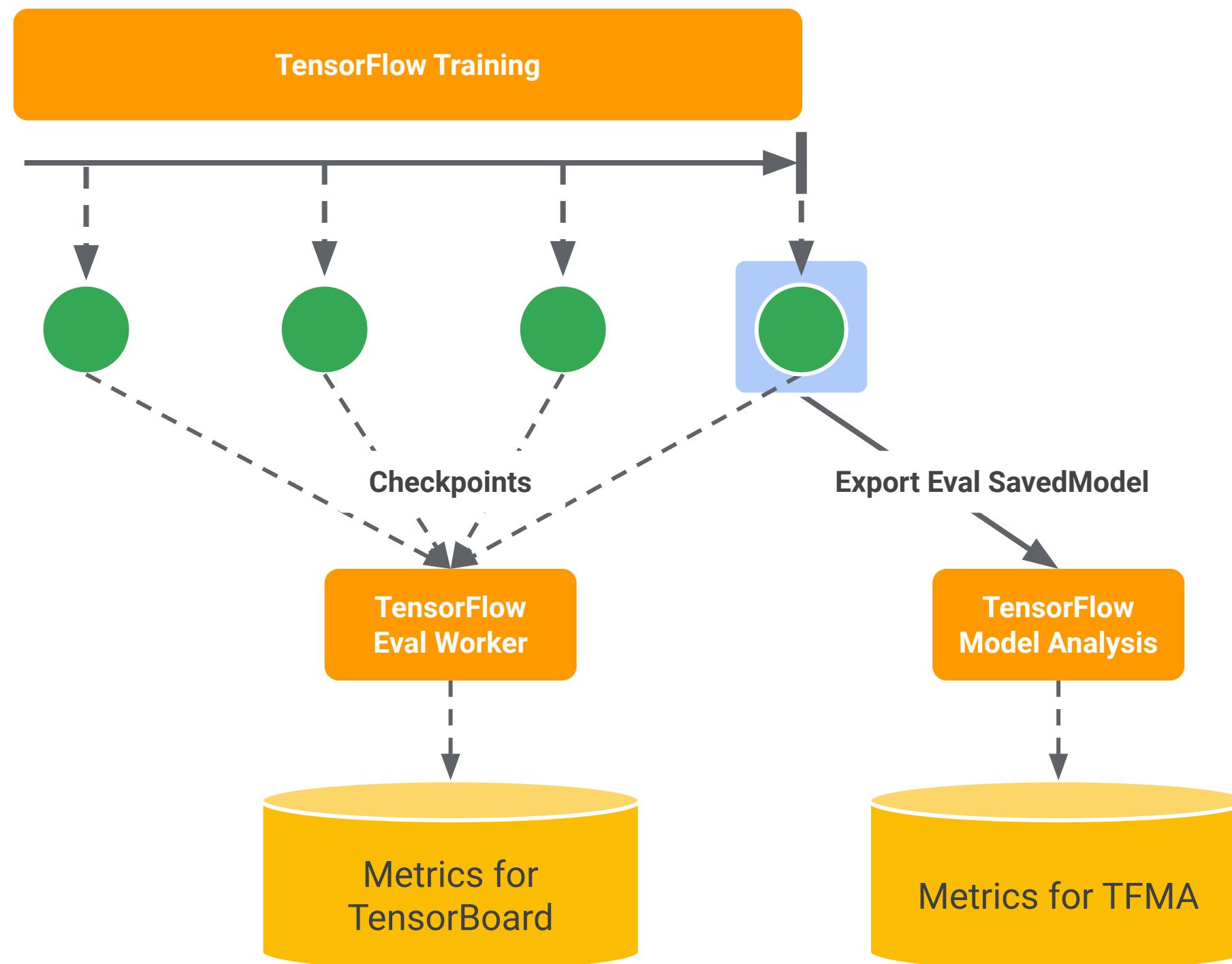
## TFMA integrated Fairness Indicators for responsible AI development



## TFMA scales model evaluation with Apache Beam



# How does TFMA compare to TensorBoard?



---

# Lab

## TFX Walkthrough

