Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Introduction**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l1_introduction

# Designing High Performance ML Systems

Laurence Moroney

Title Safe >

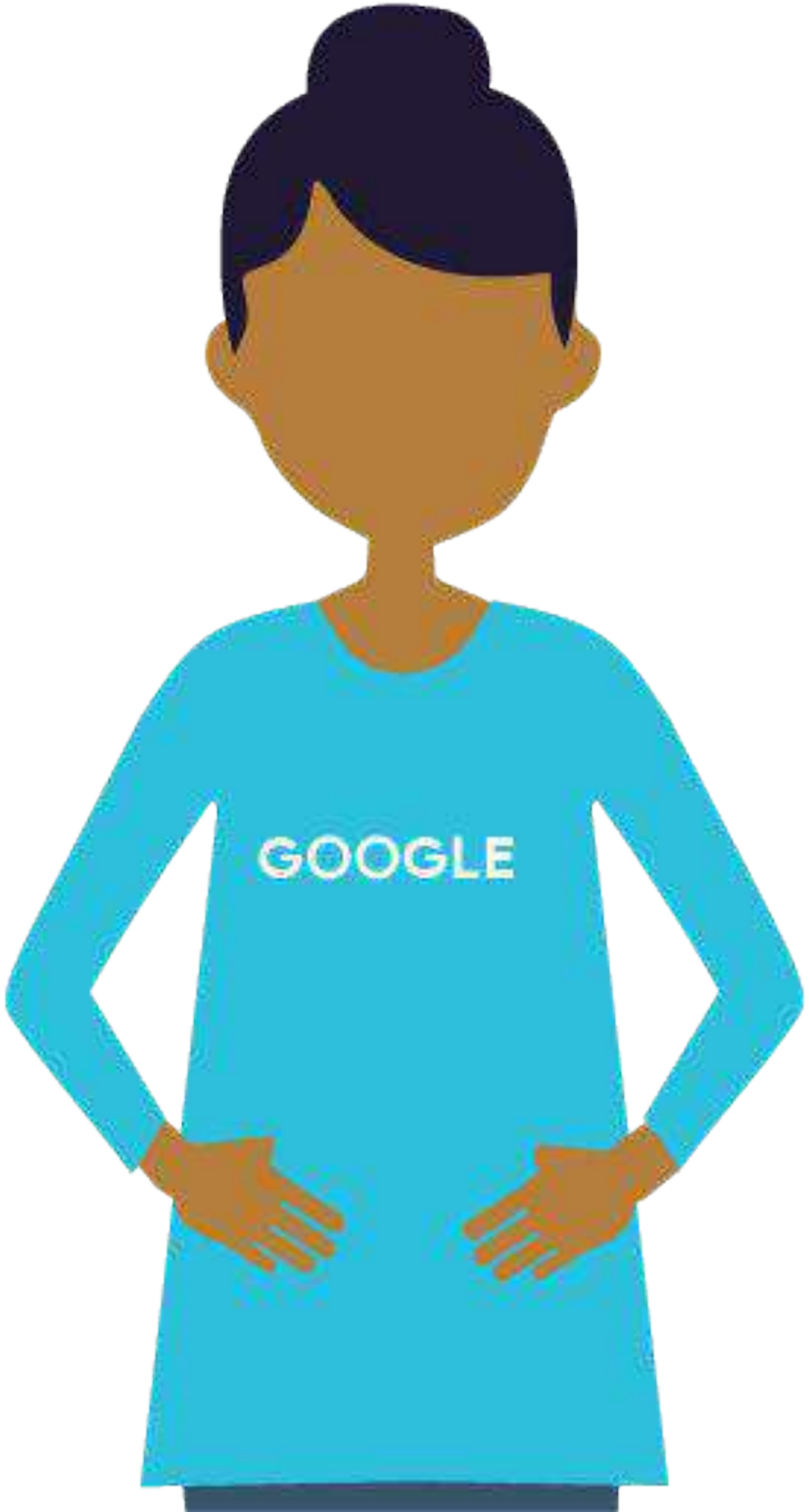
< Action Safe

# Learn how to...

**Identify performance considerations for ML models**

Choose appropriate ML infrastructure

Select a distribution strategy

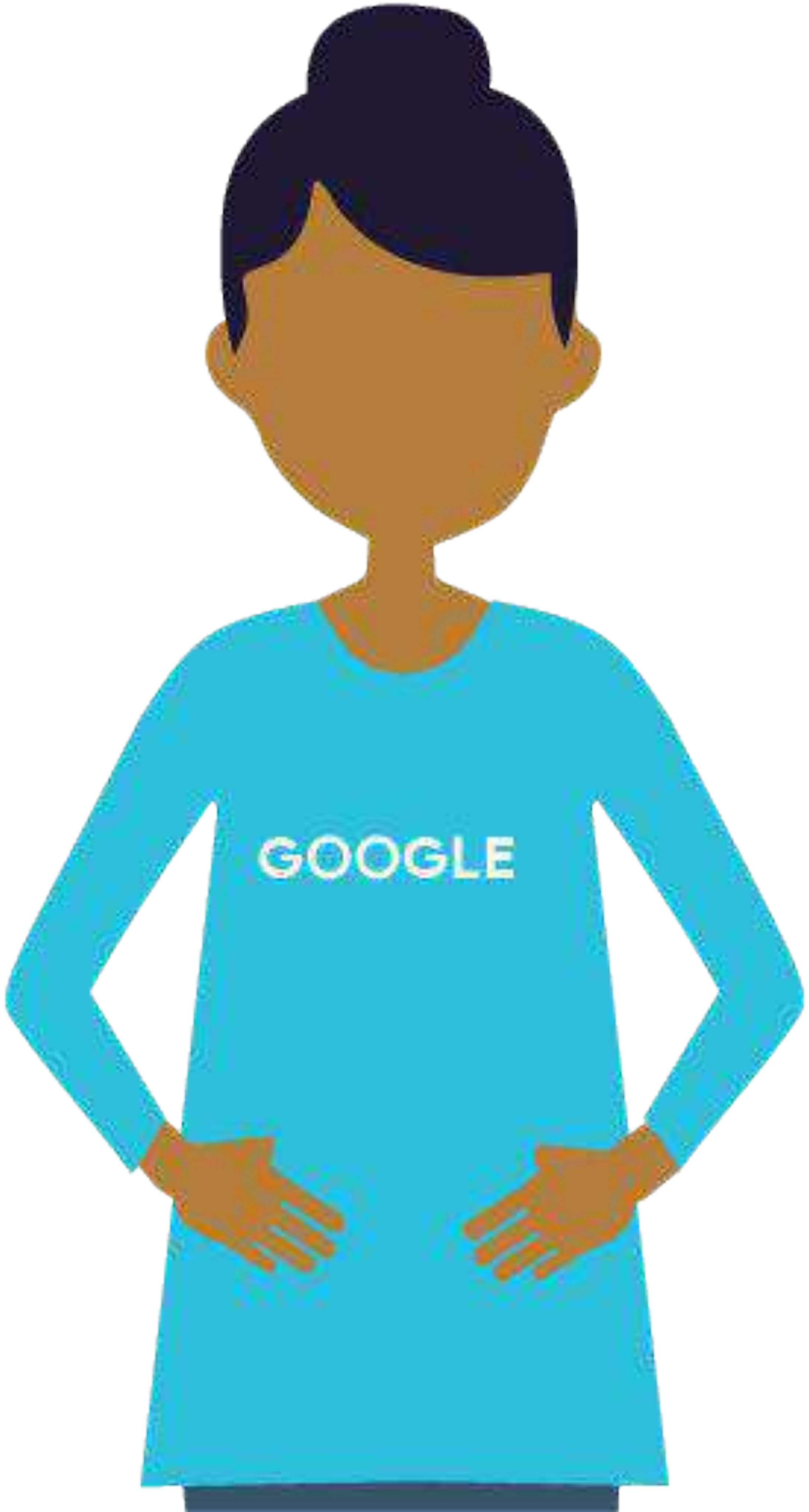


# Learn how to...

Identify performance considerations for ML models

**Choose appropriate ML infrastructure**

Select a distribution strategy

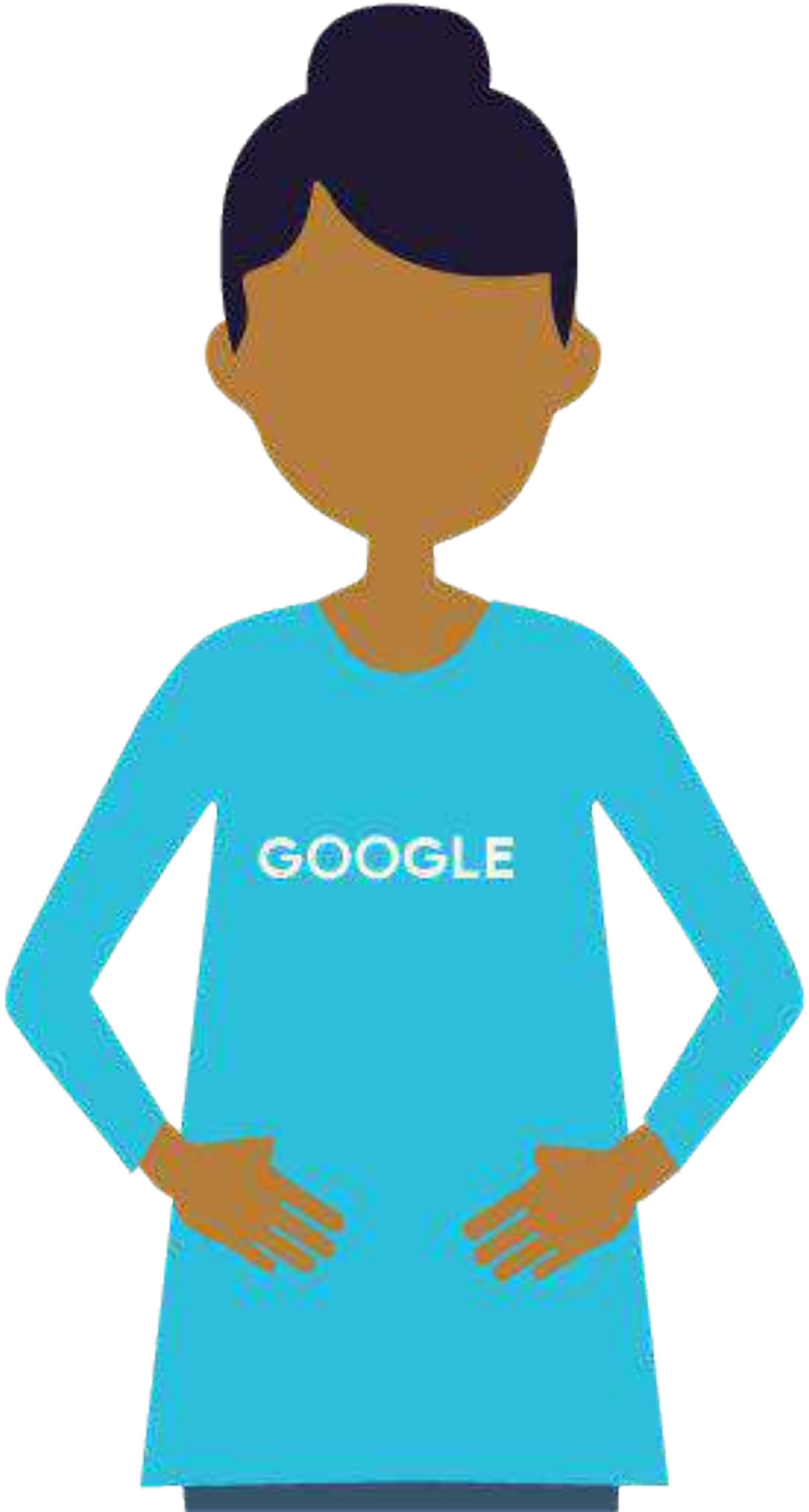


# Learn how to...

Identify performance considerations for ML models

Choose appropriate ML infrastructure

**Select a distribution strategy**



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Aspects of performance: Training**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l2_aspects_of_performance:_training

# Agenda

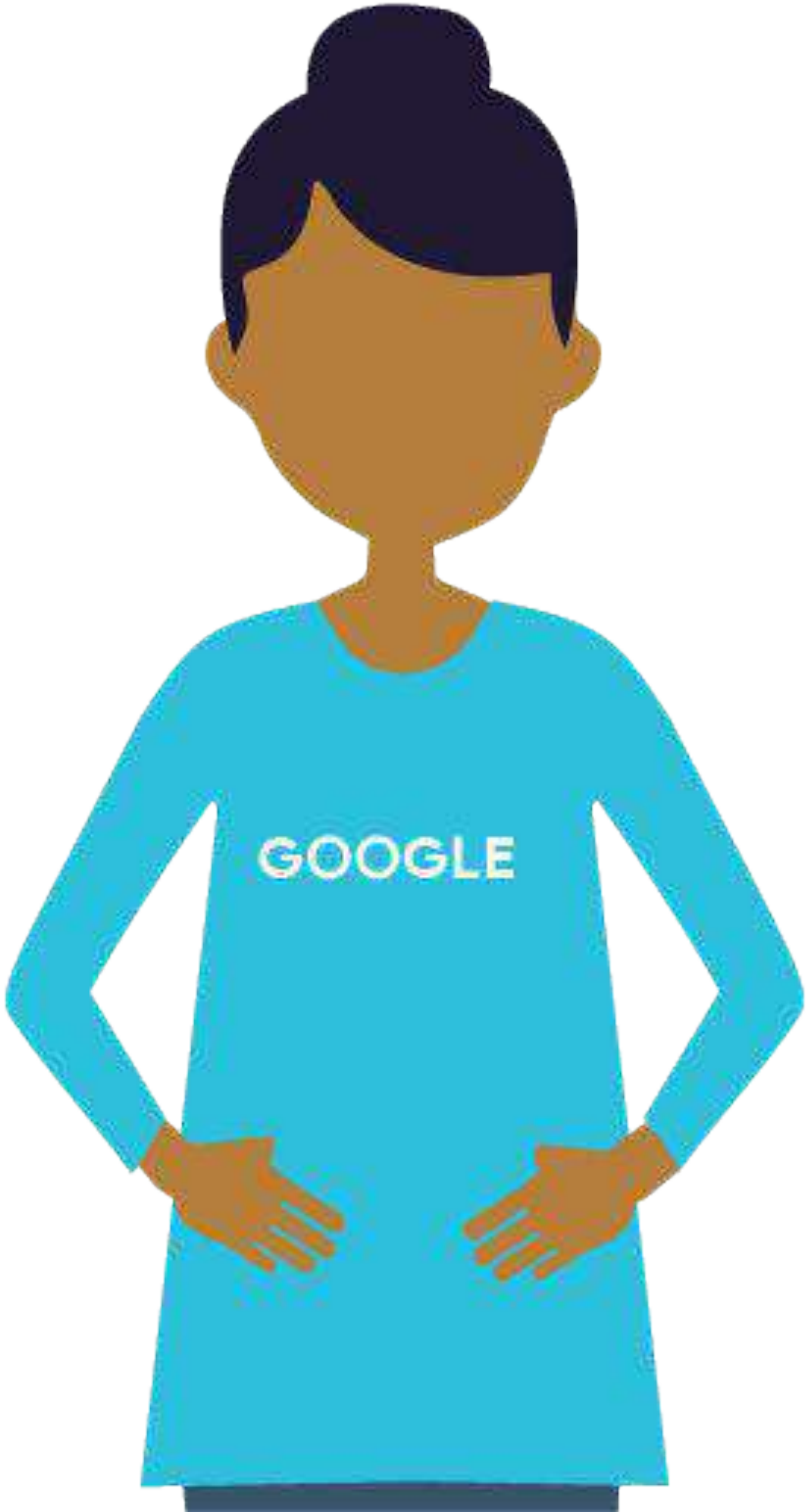
**Distributed training**

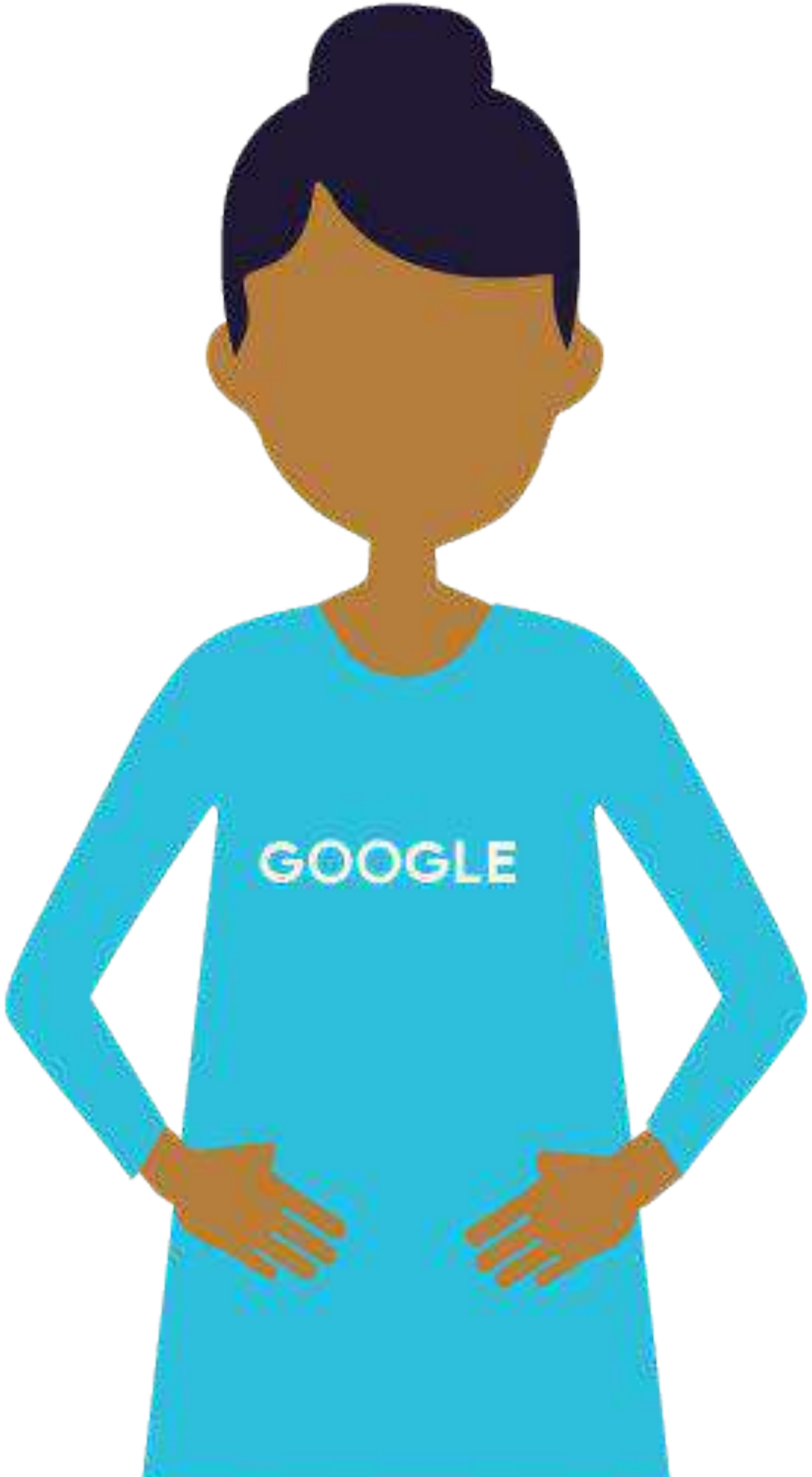
Faster input pipelines

Data parallelism (All Reduce)

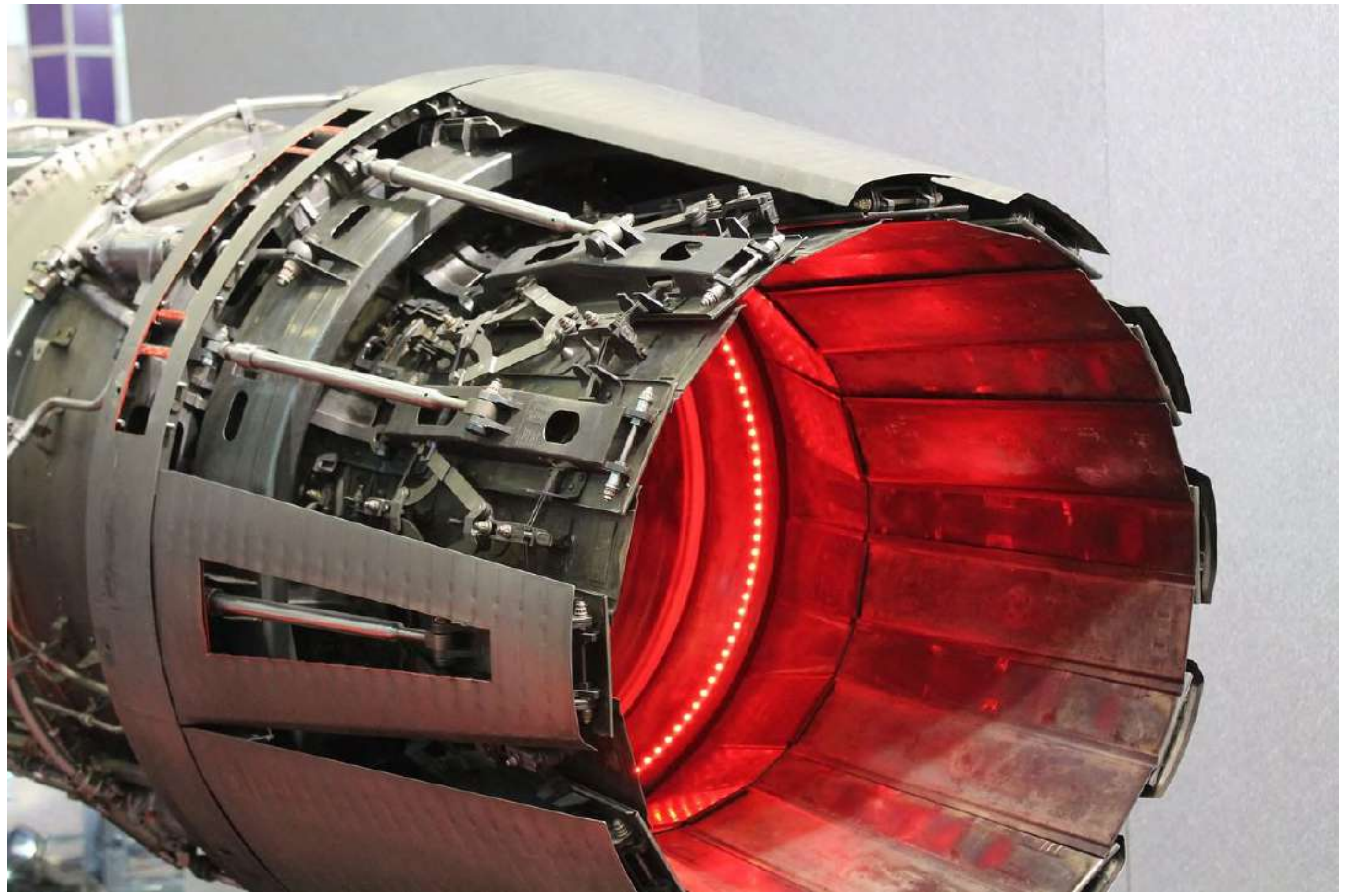
Parameter Server approach

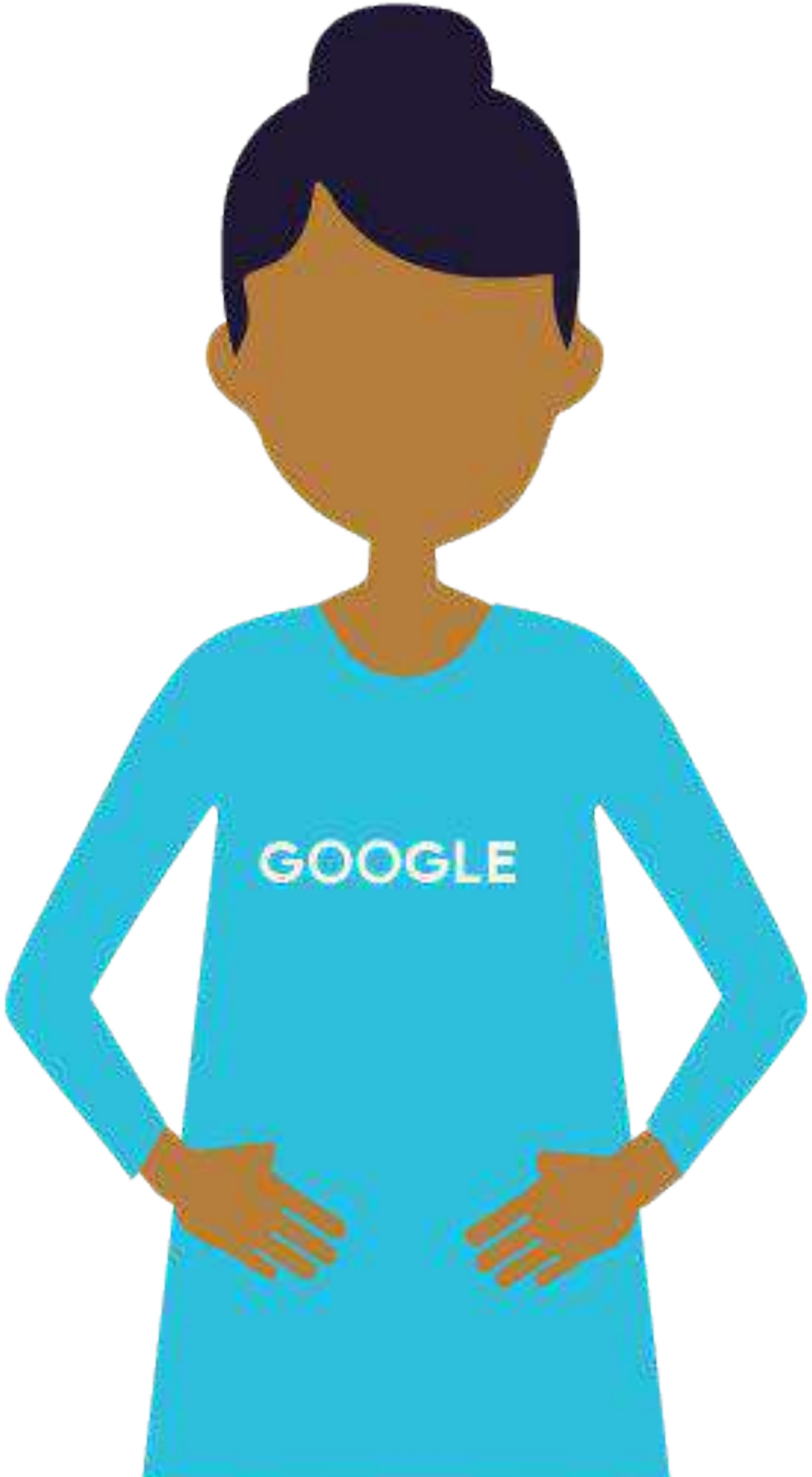
Inference



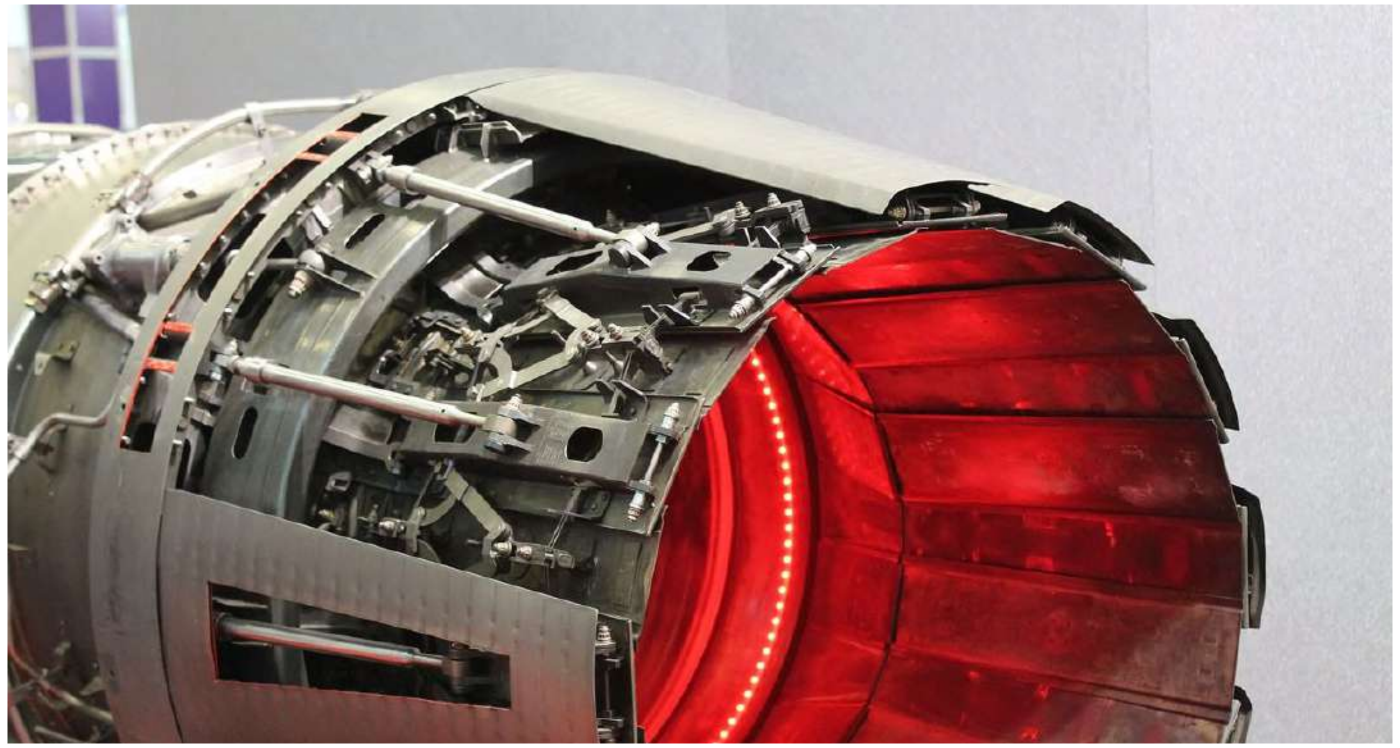


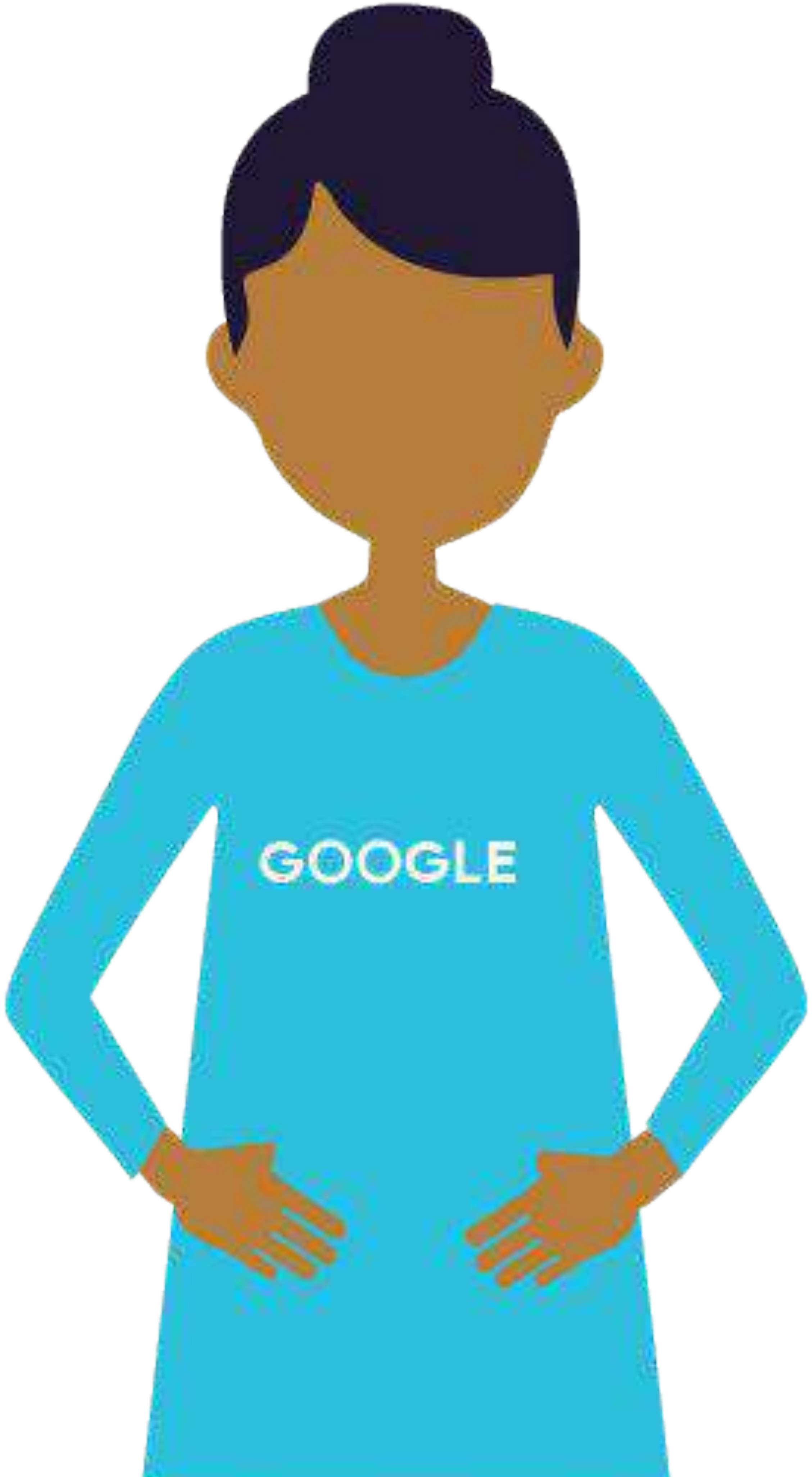

High Performance ML



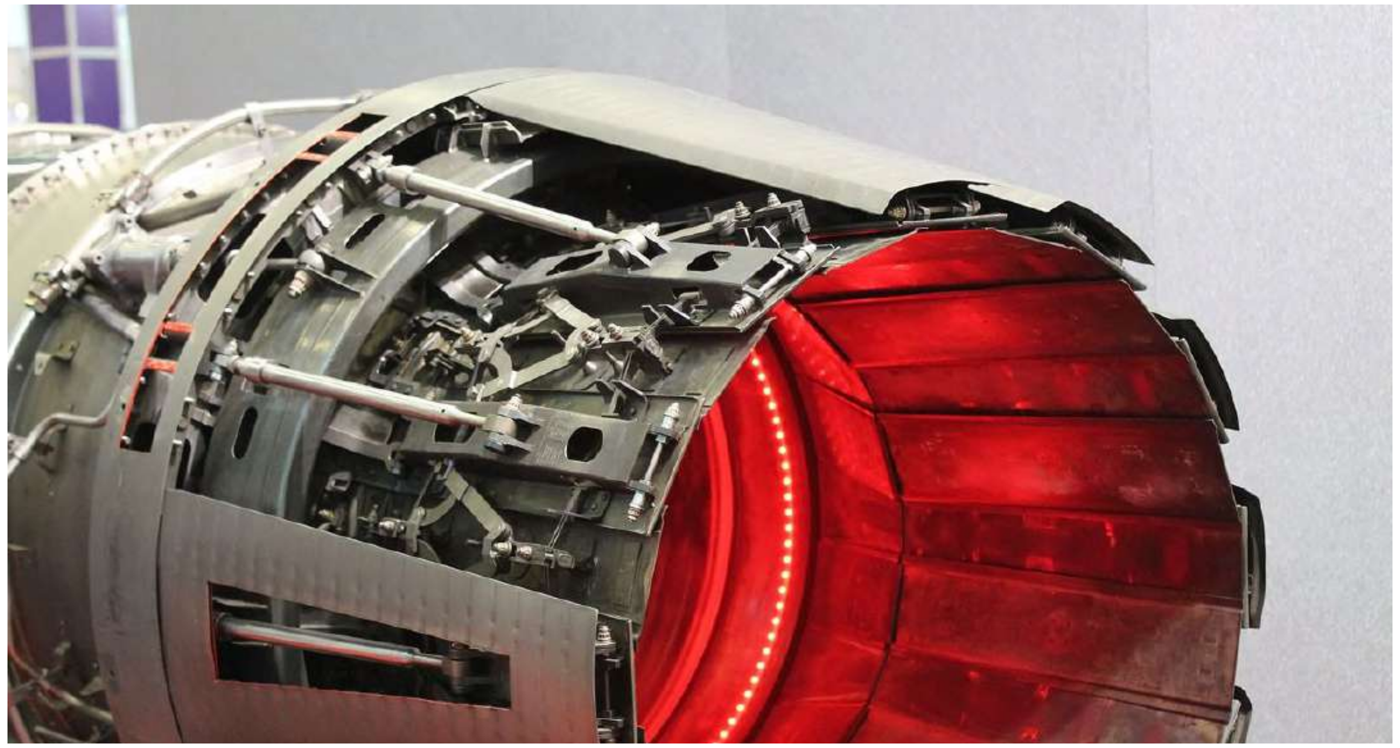


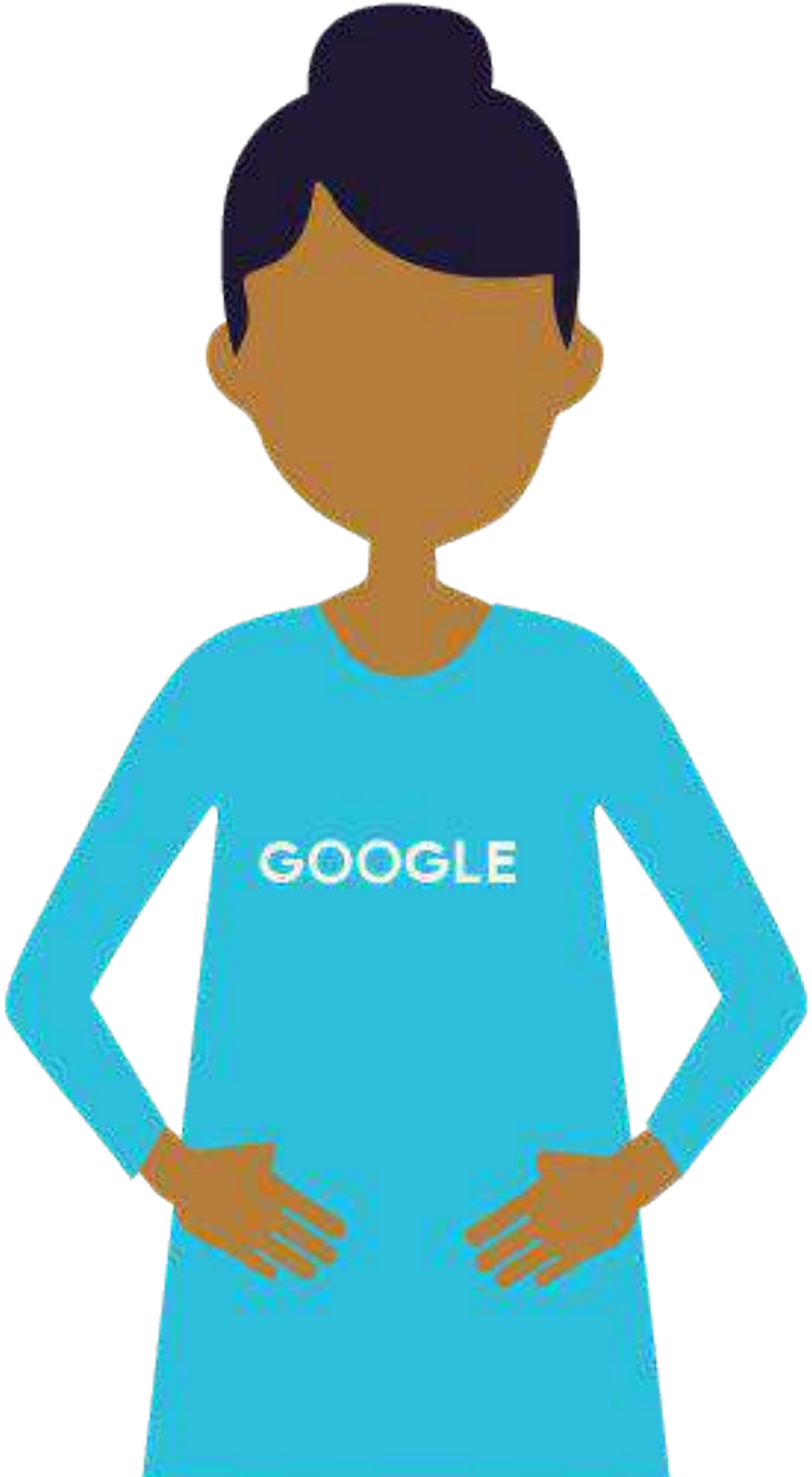

# High Performance ML



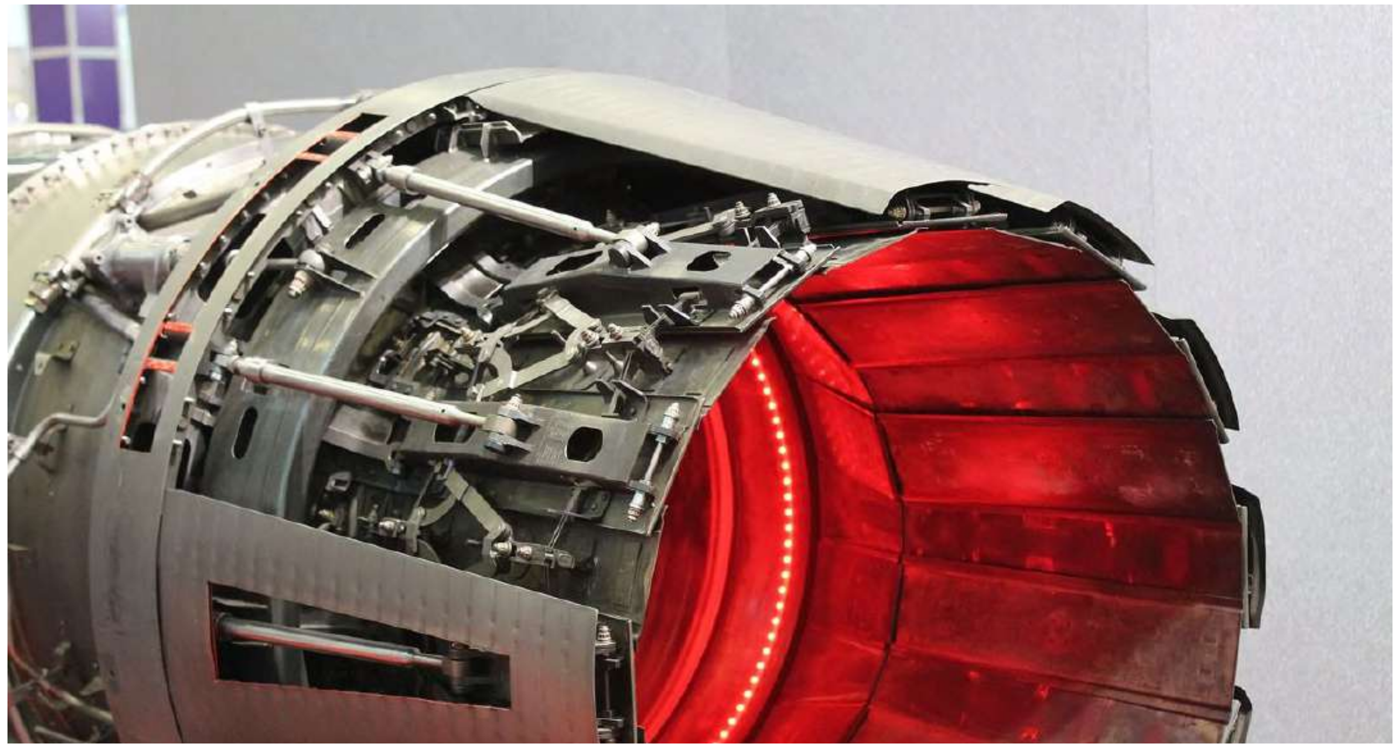


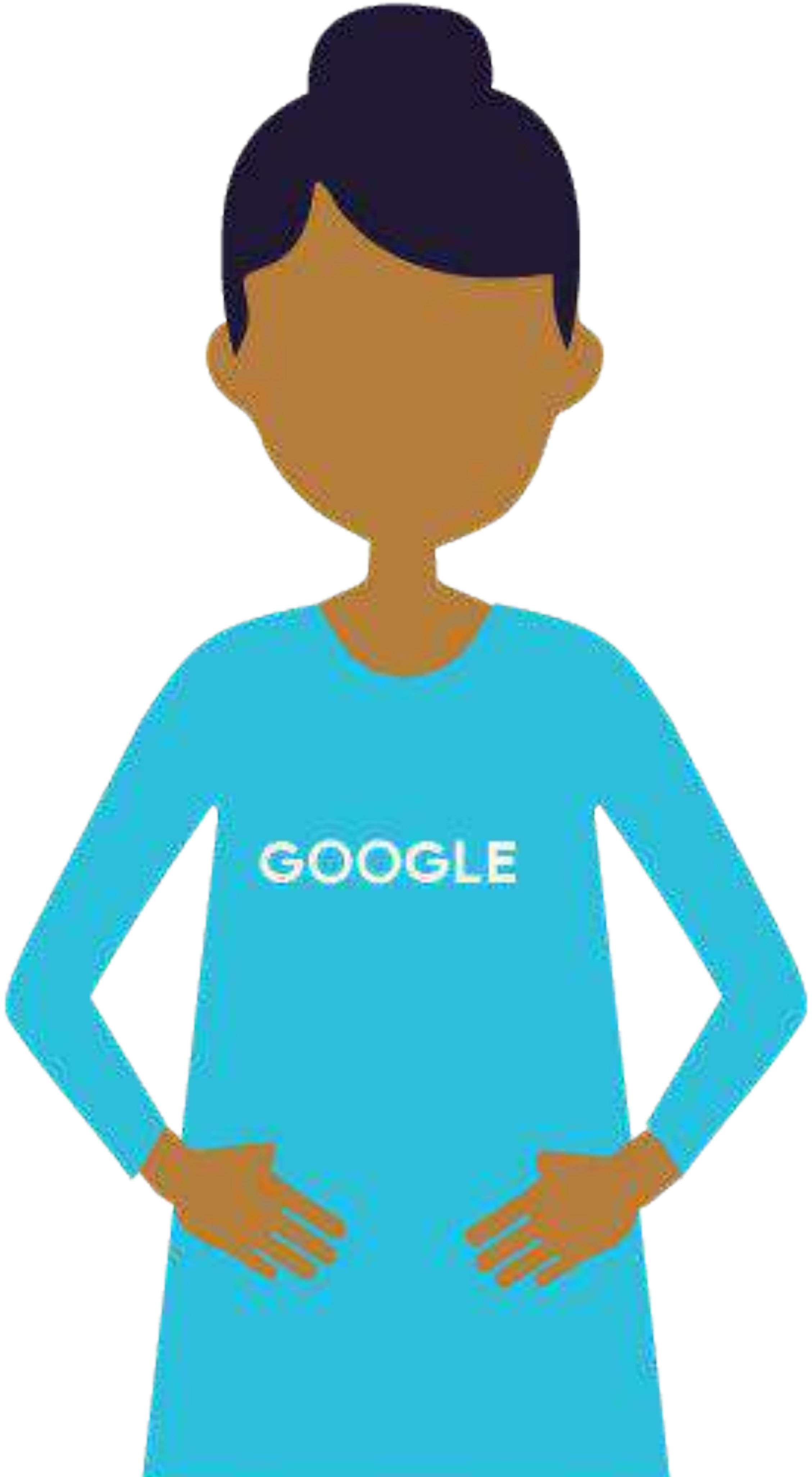

# High Performance ML



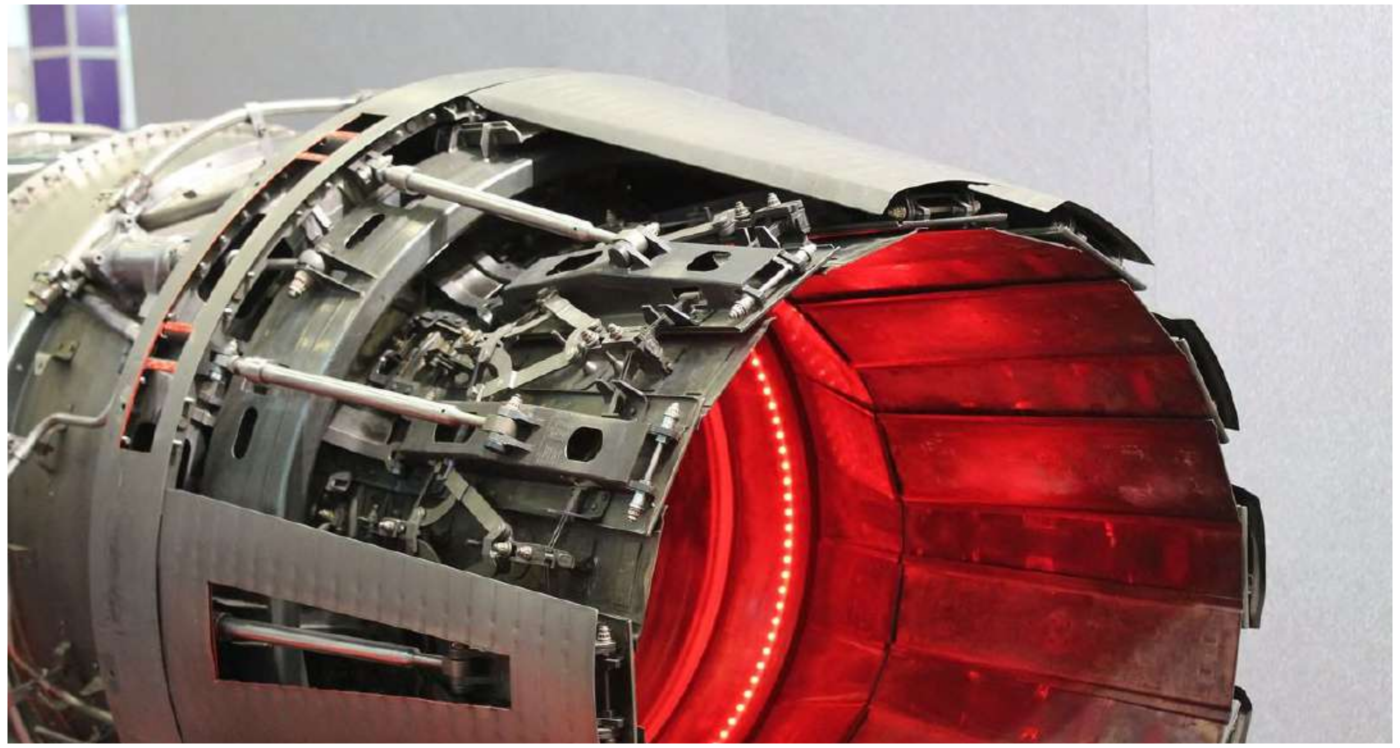


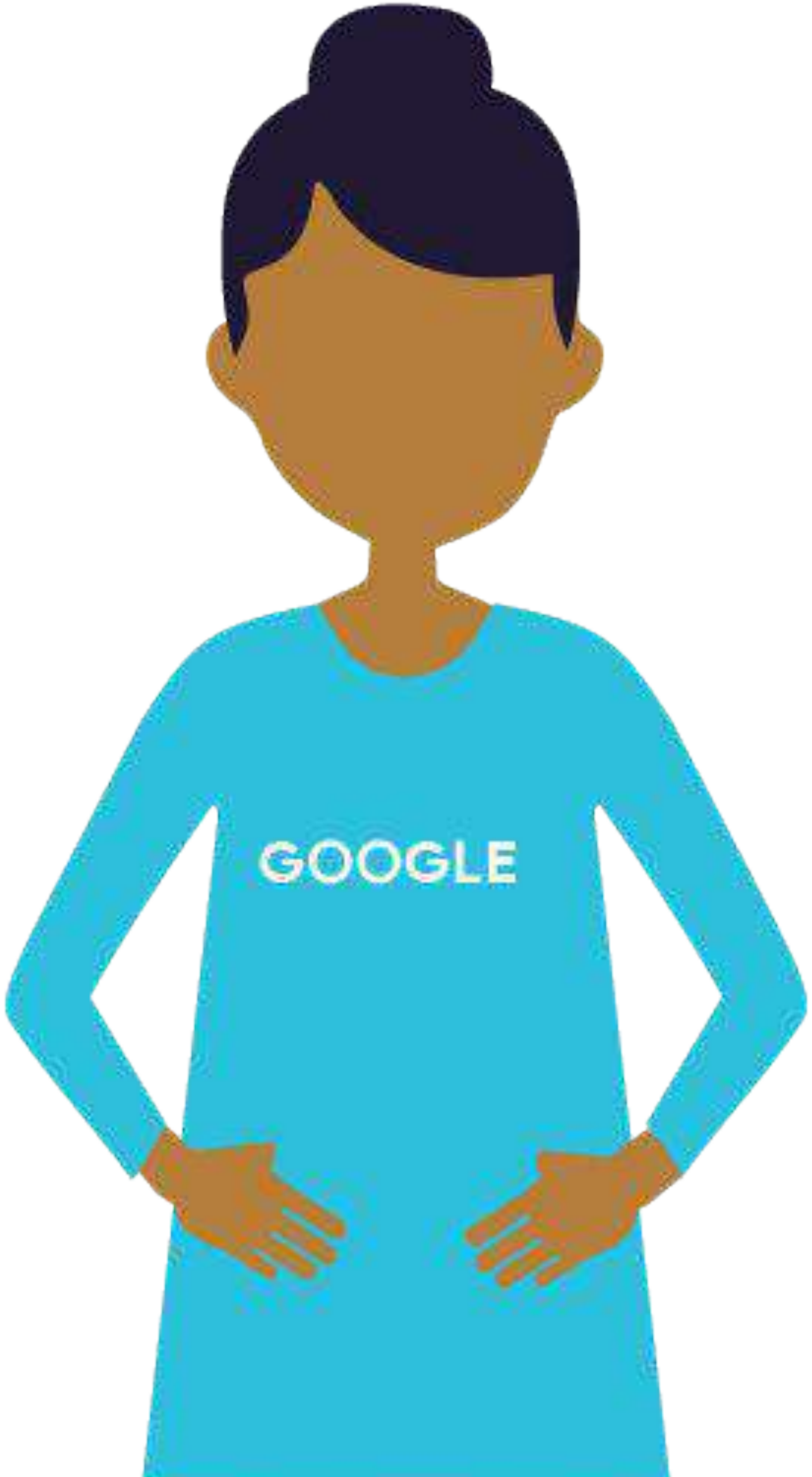

# High Performance ML



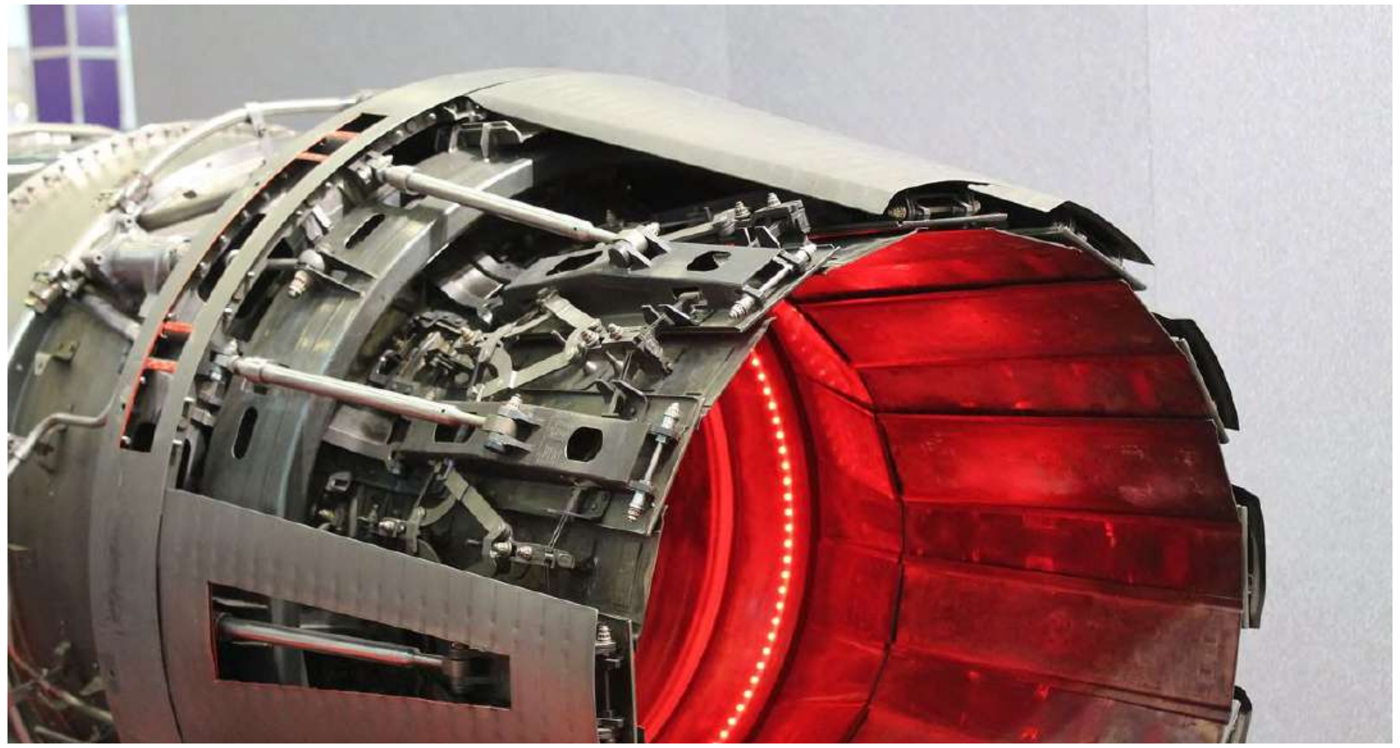



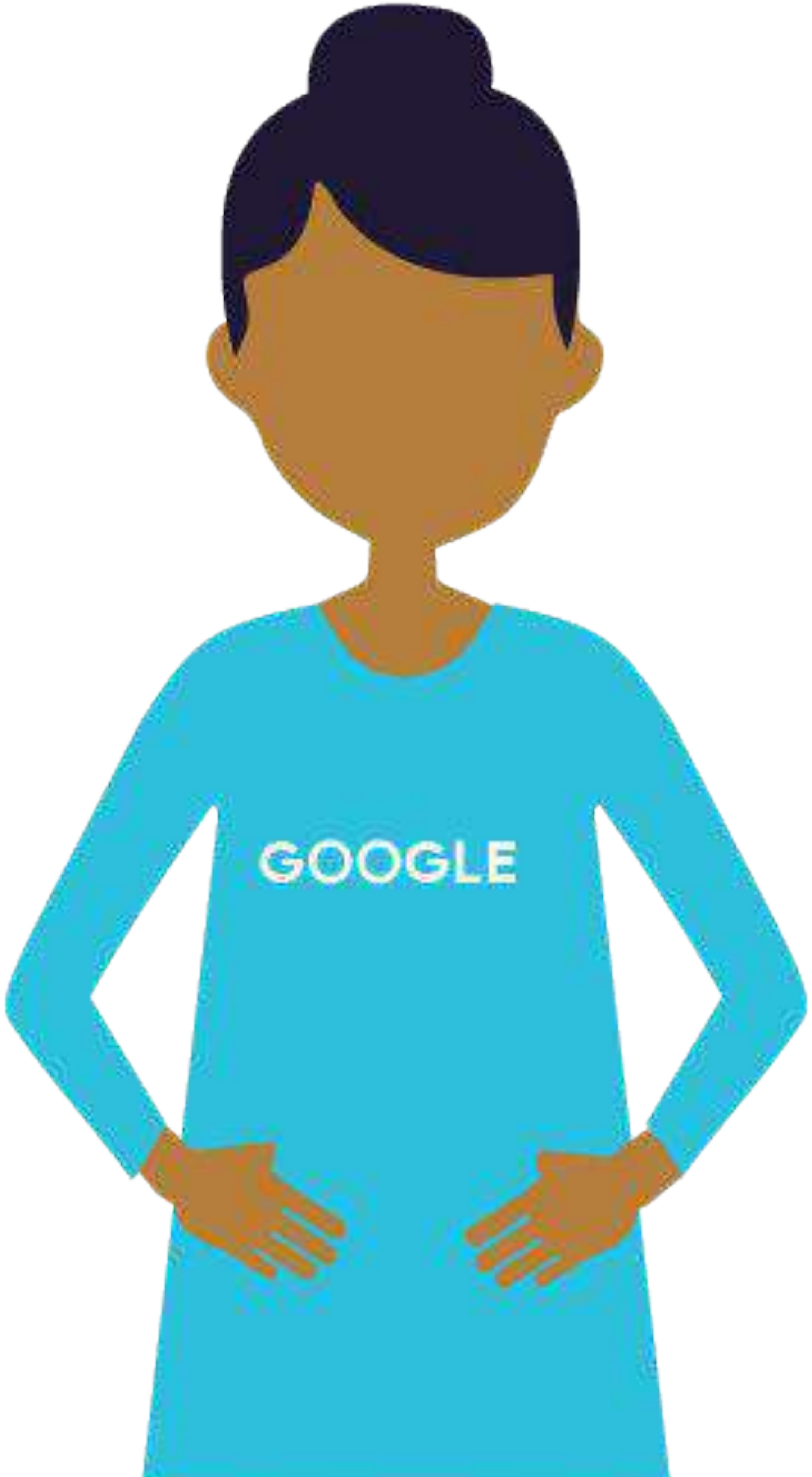

# High Performance ML




# Model Training Time



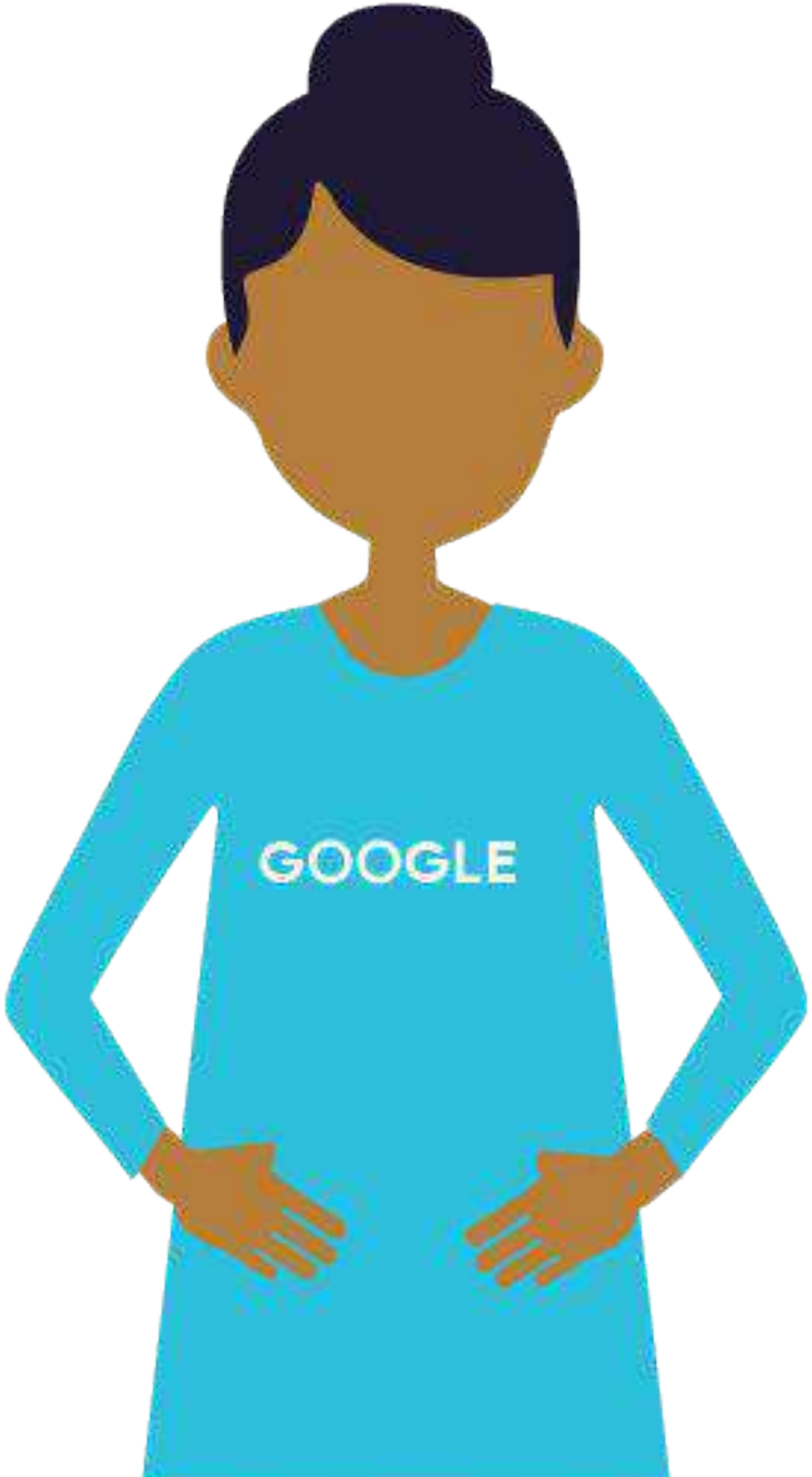



## Model Training Time



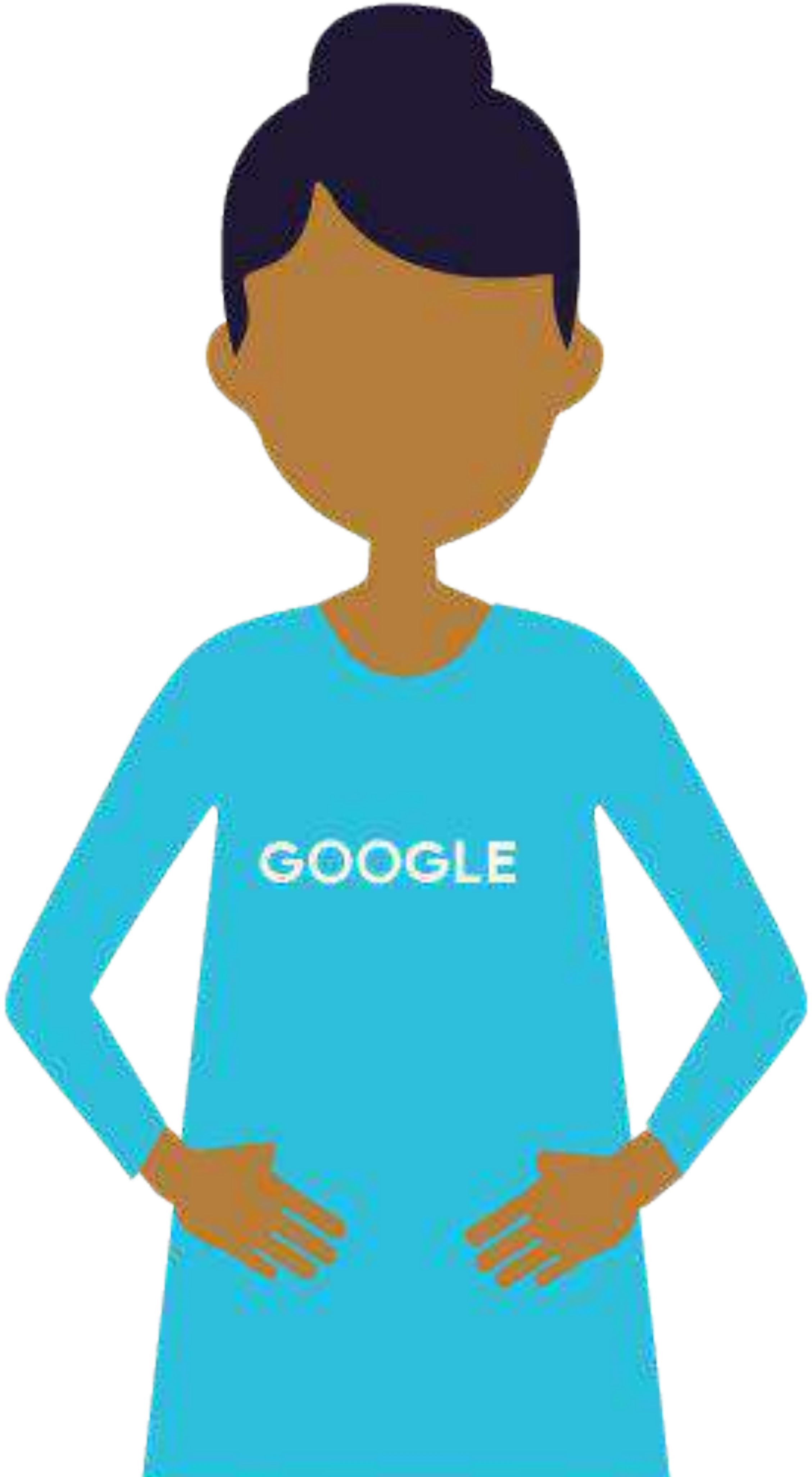




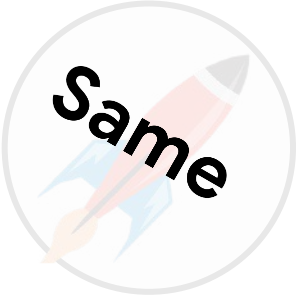

## Model Training Time



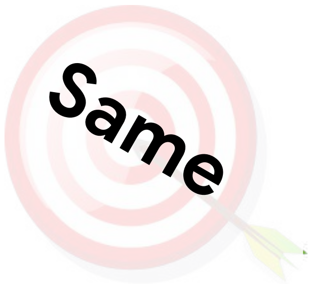



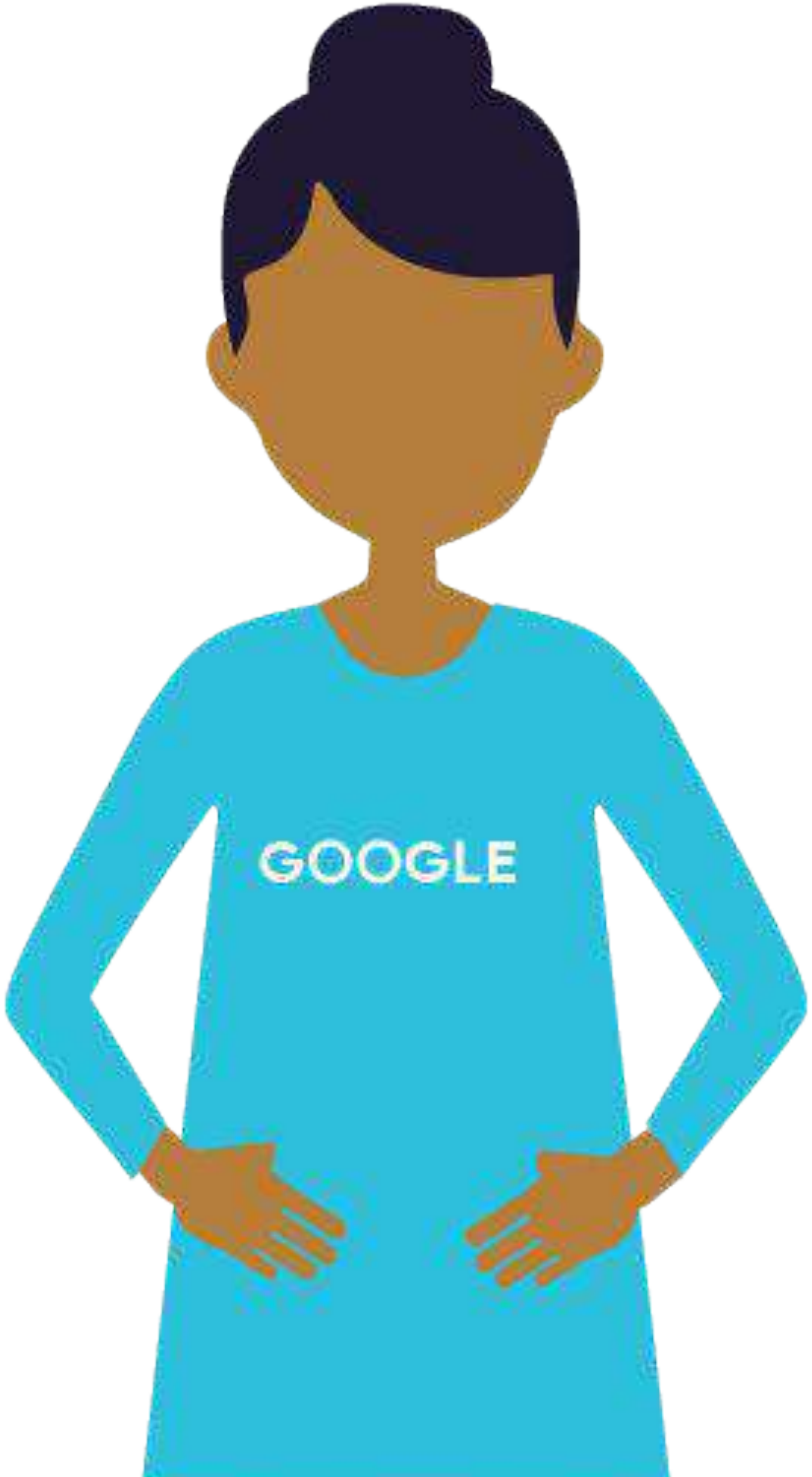

# Model Training Time



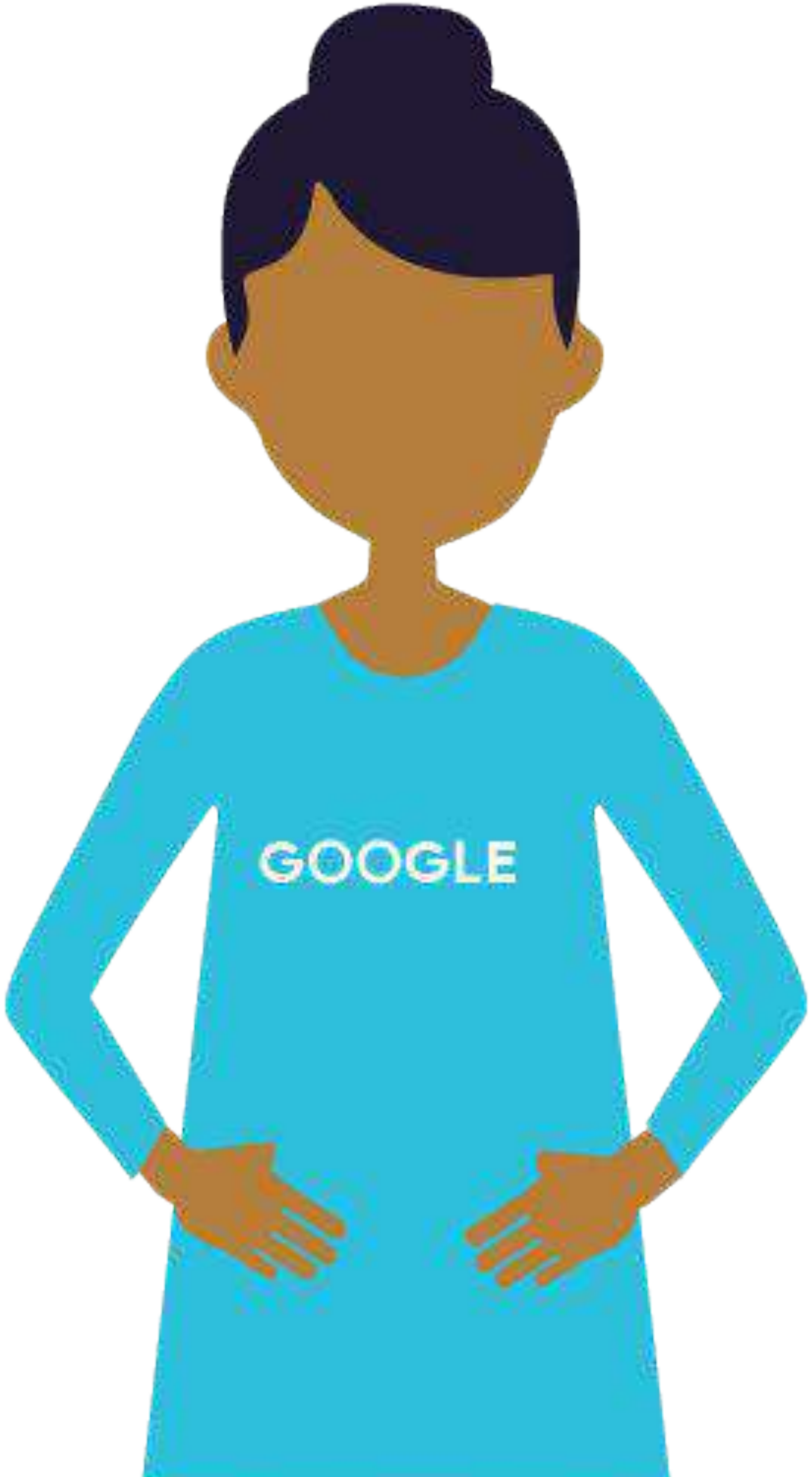





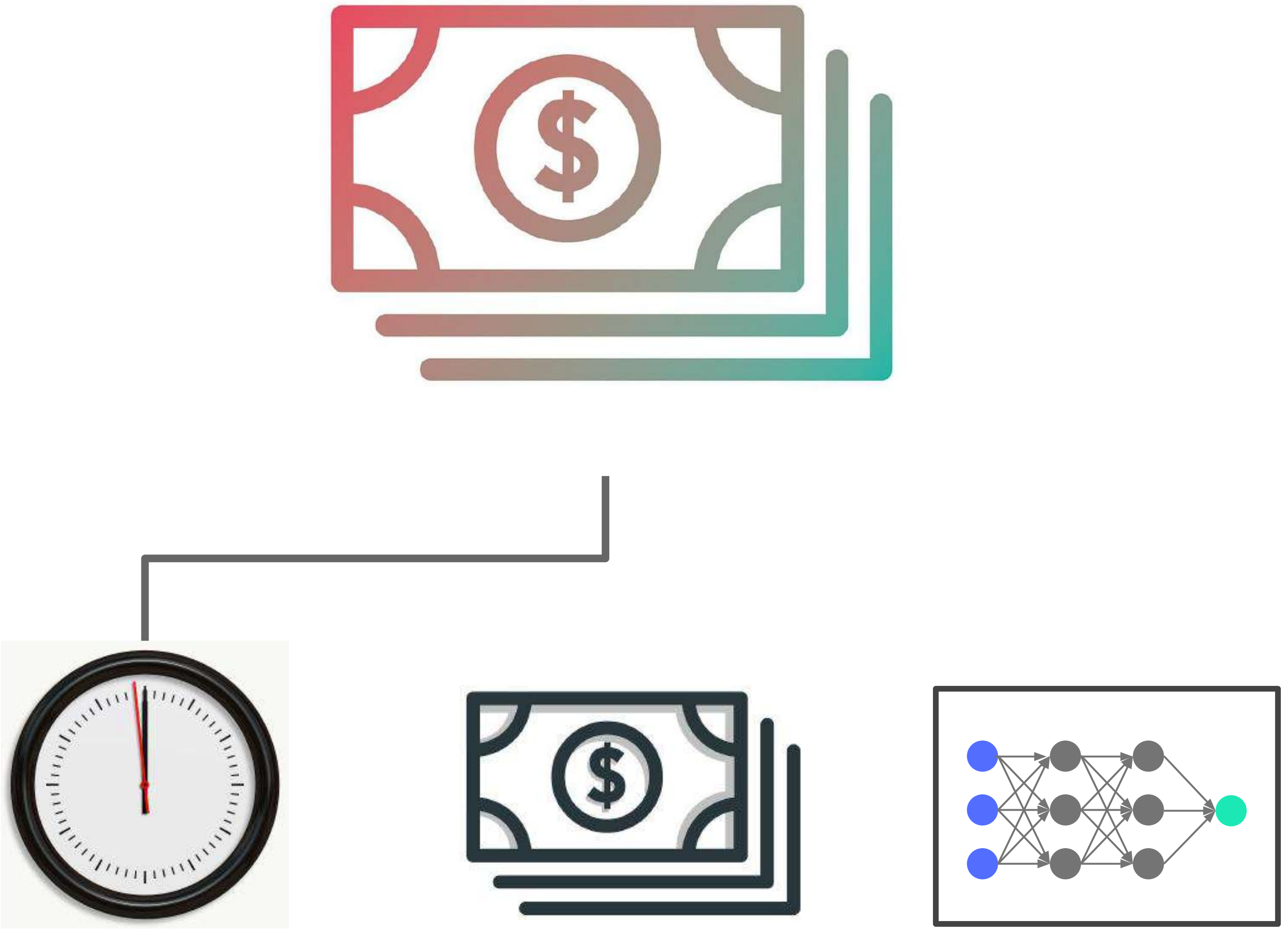


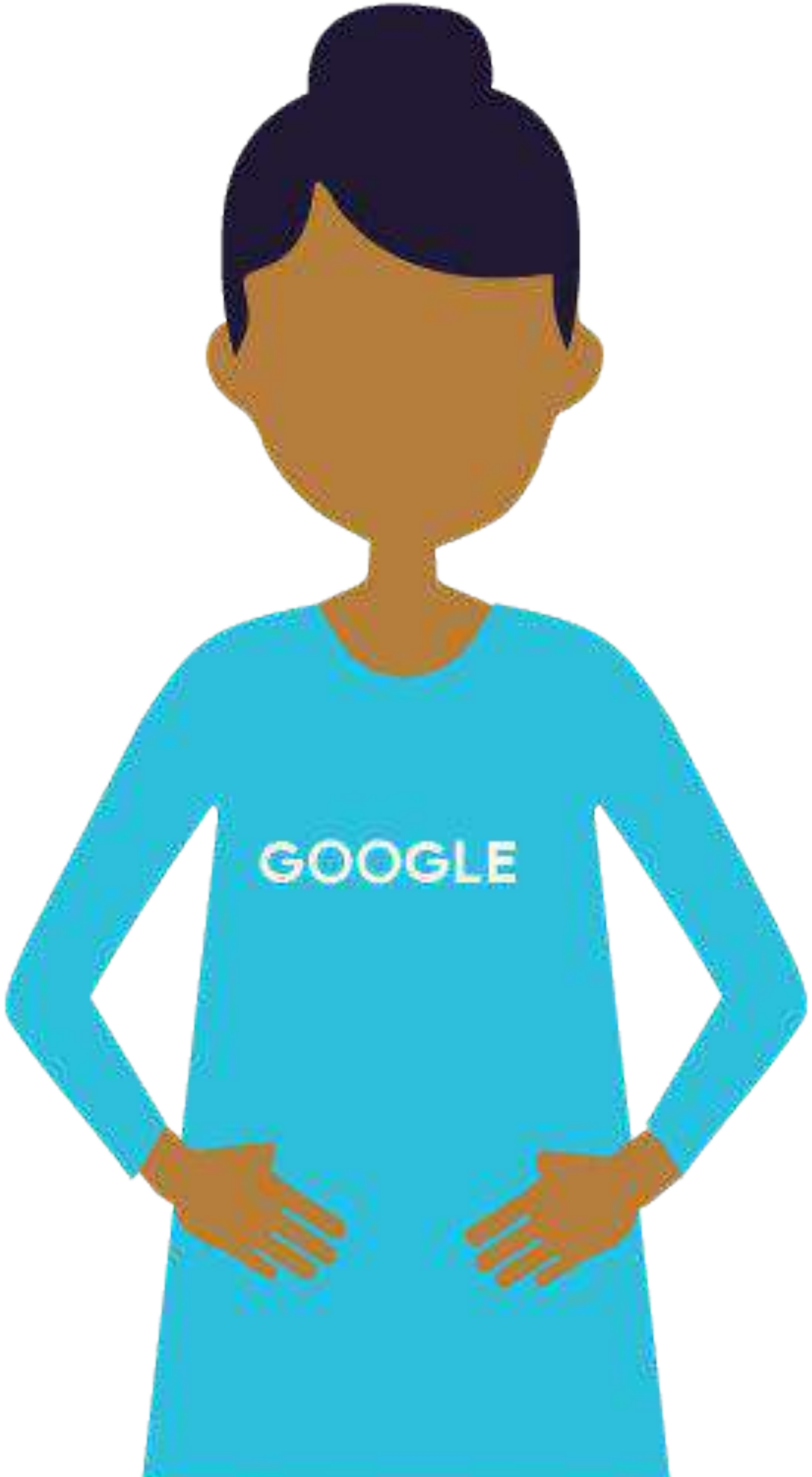



Optimizing your
Training Budget

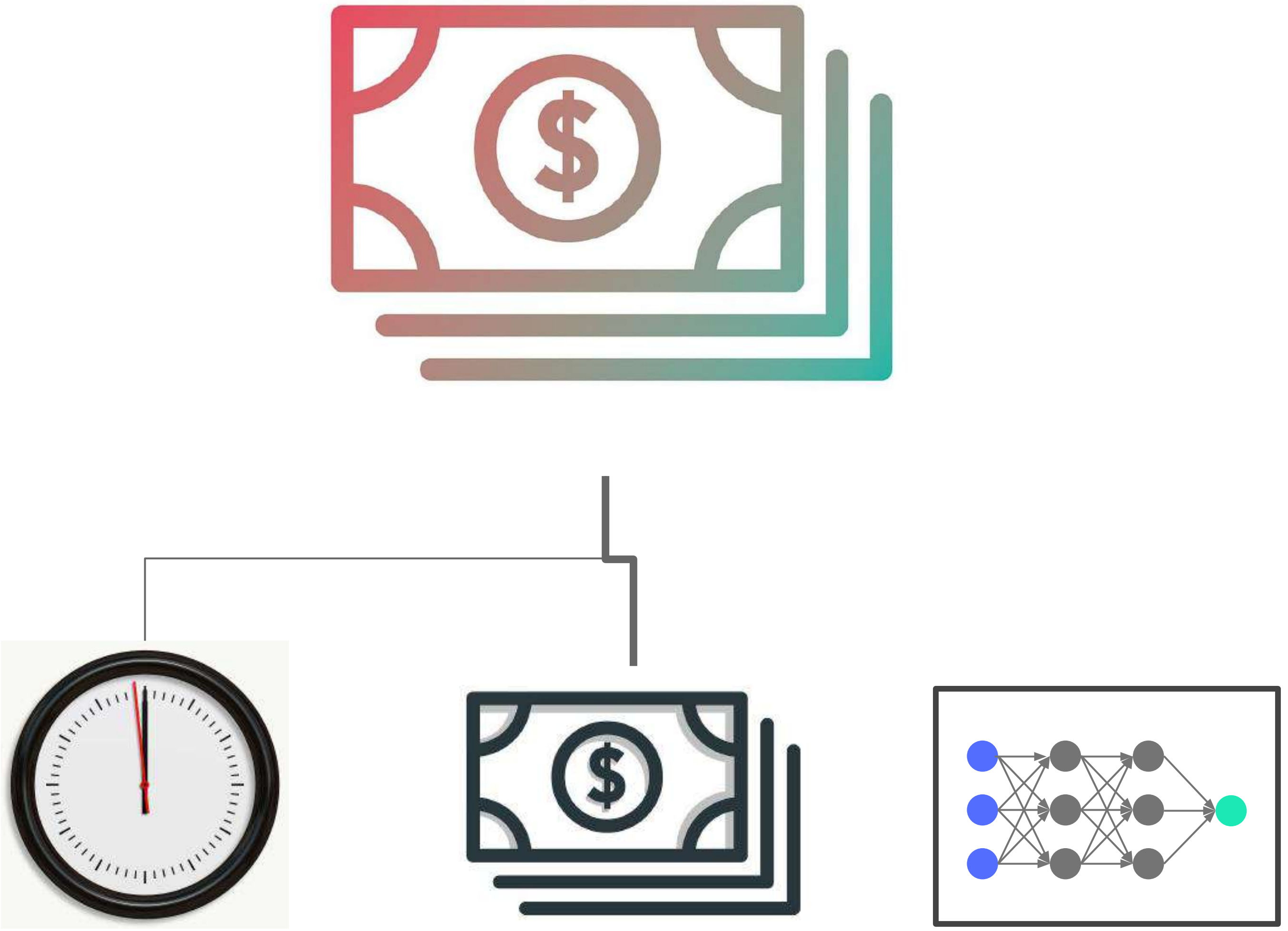


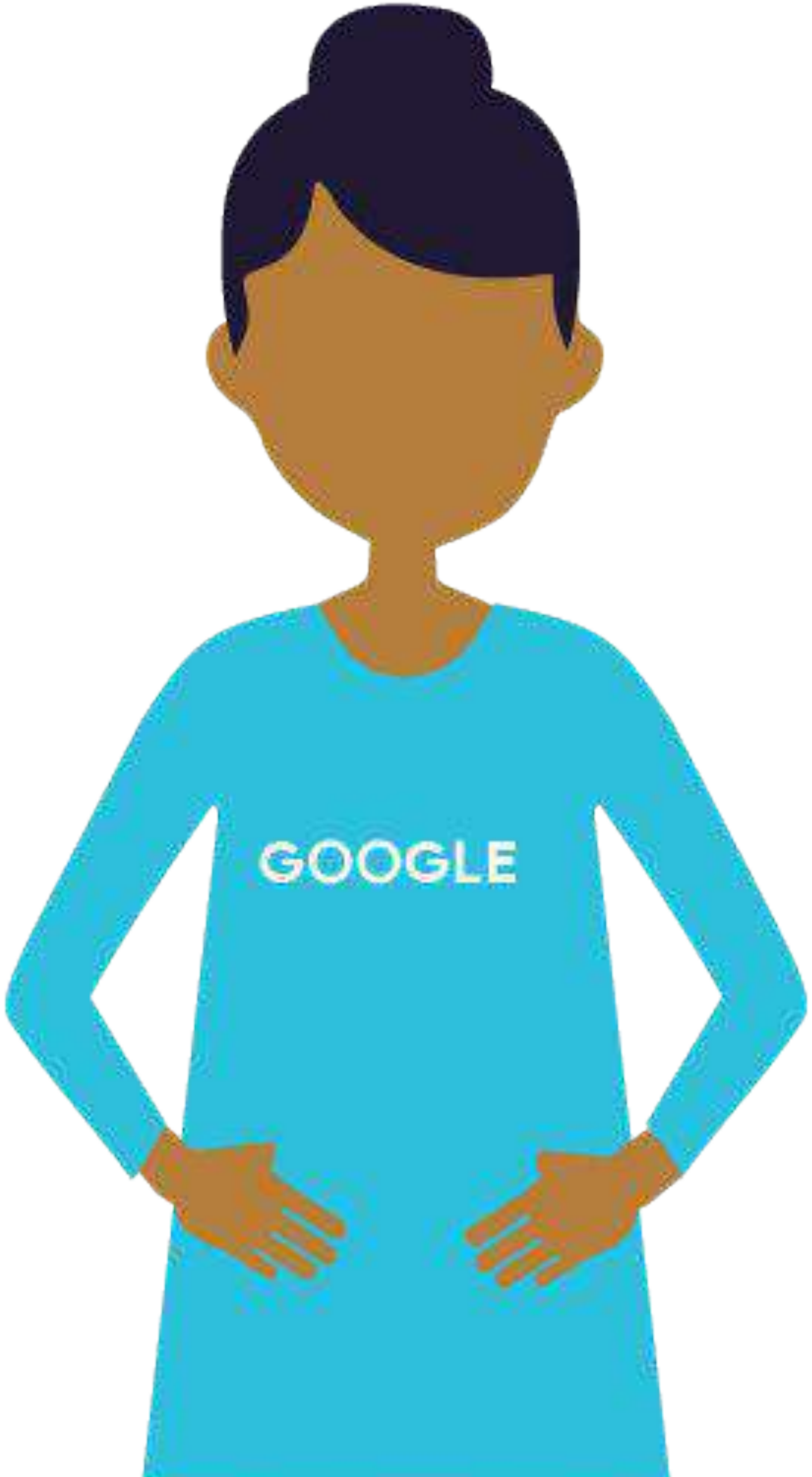

# Optimizing your Training Budget



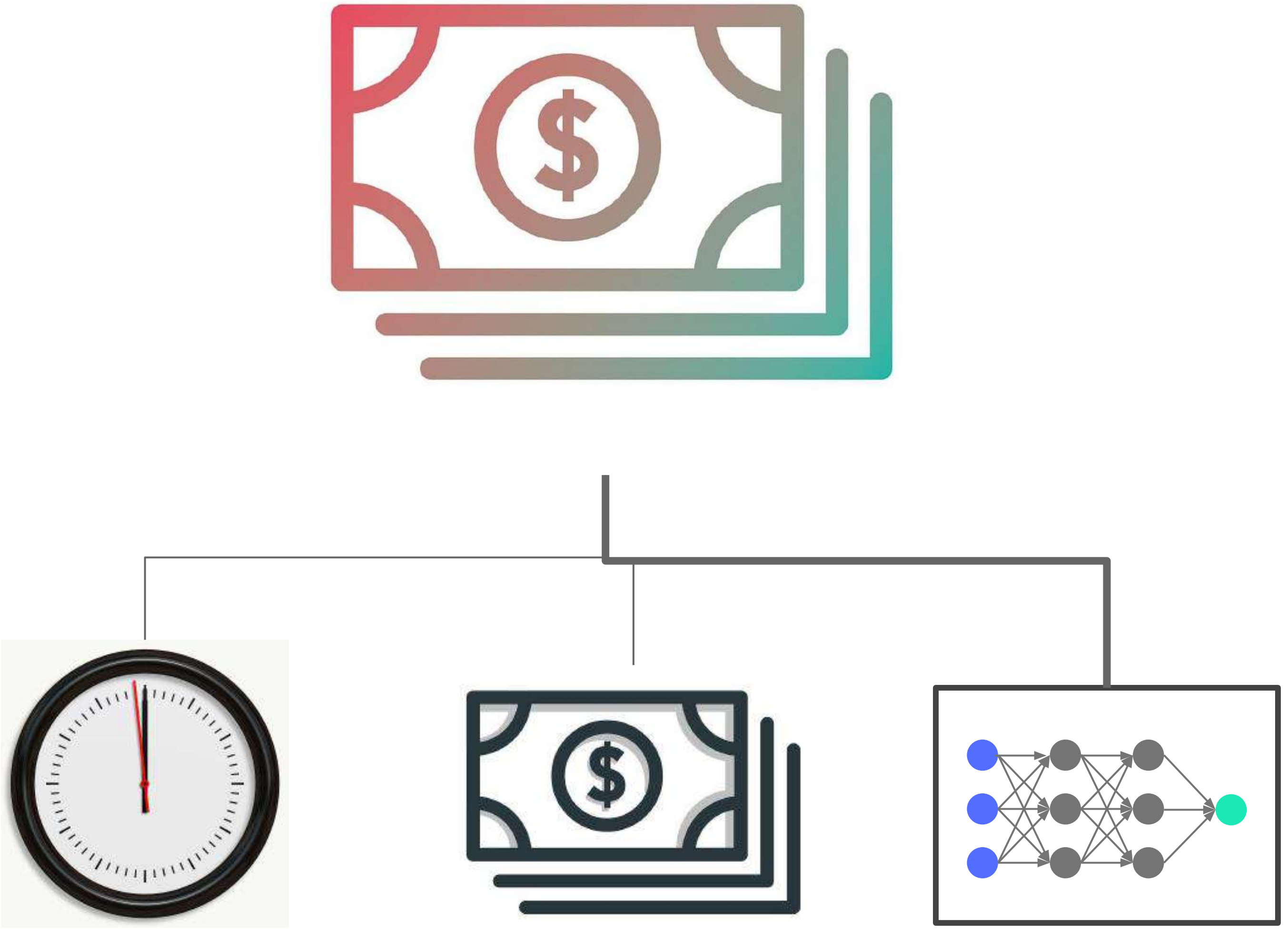



# Optimizing your Training Budget

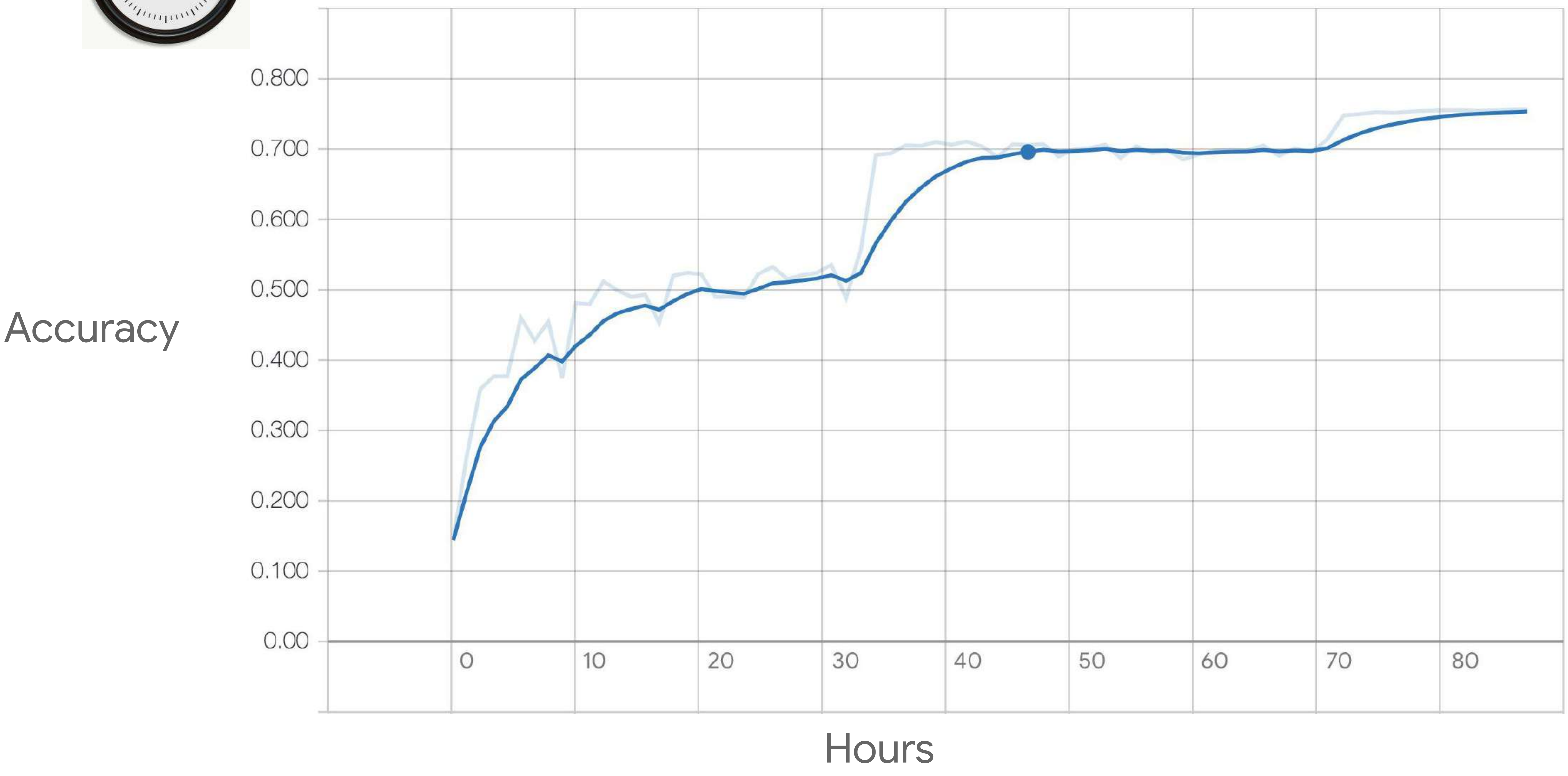




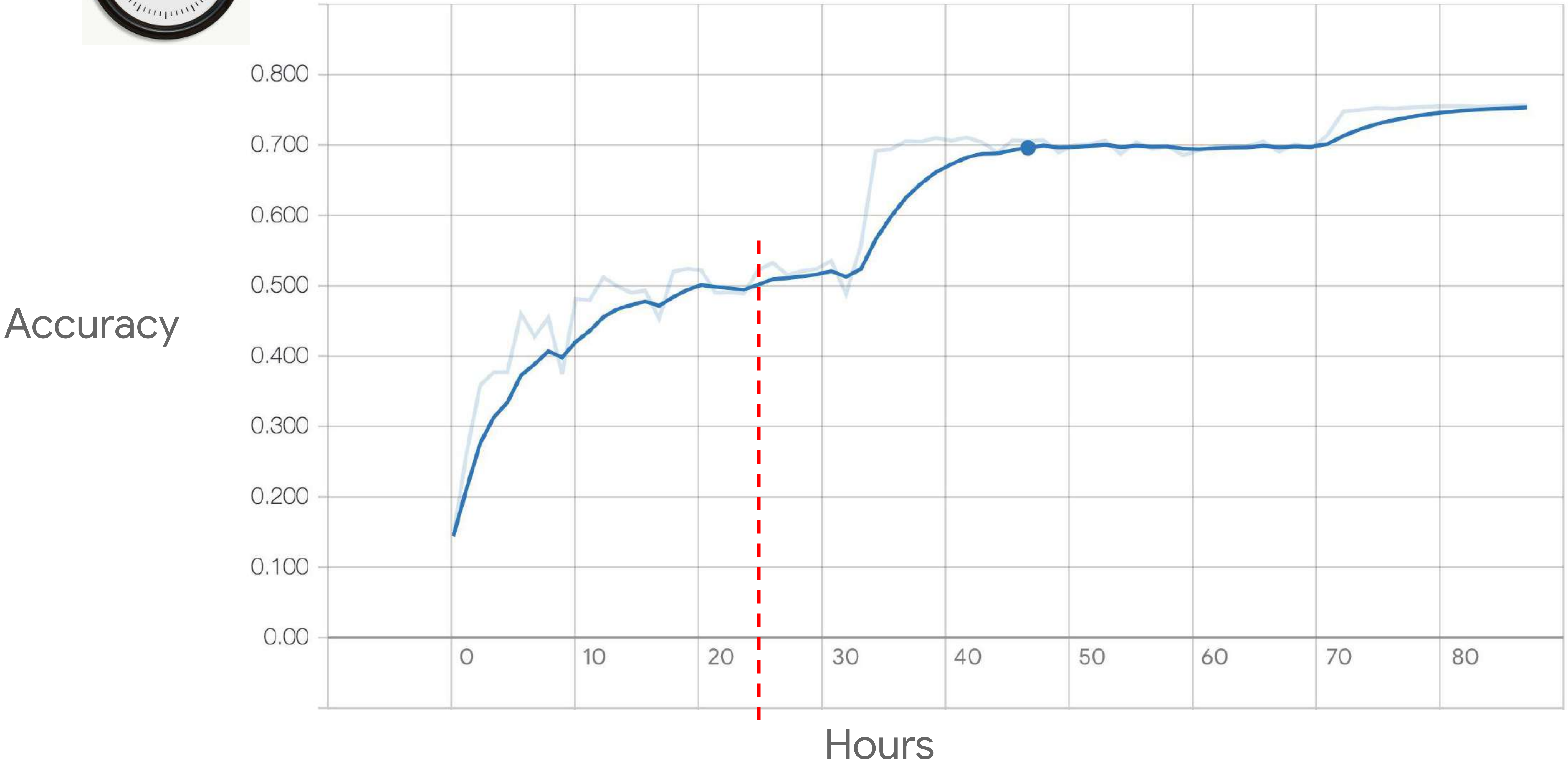

# Optimizing your Training Budget



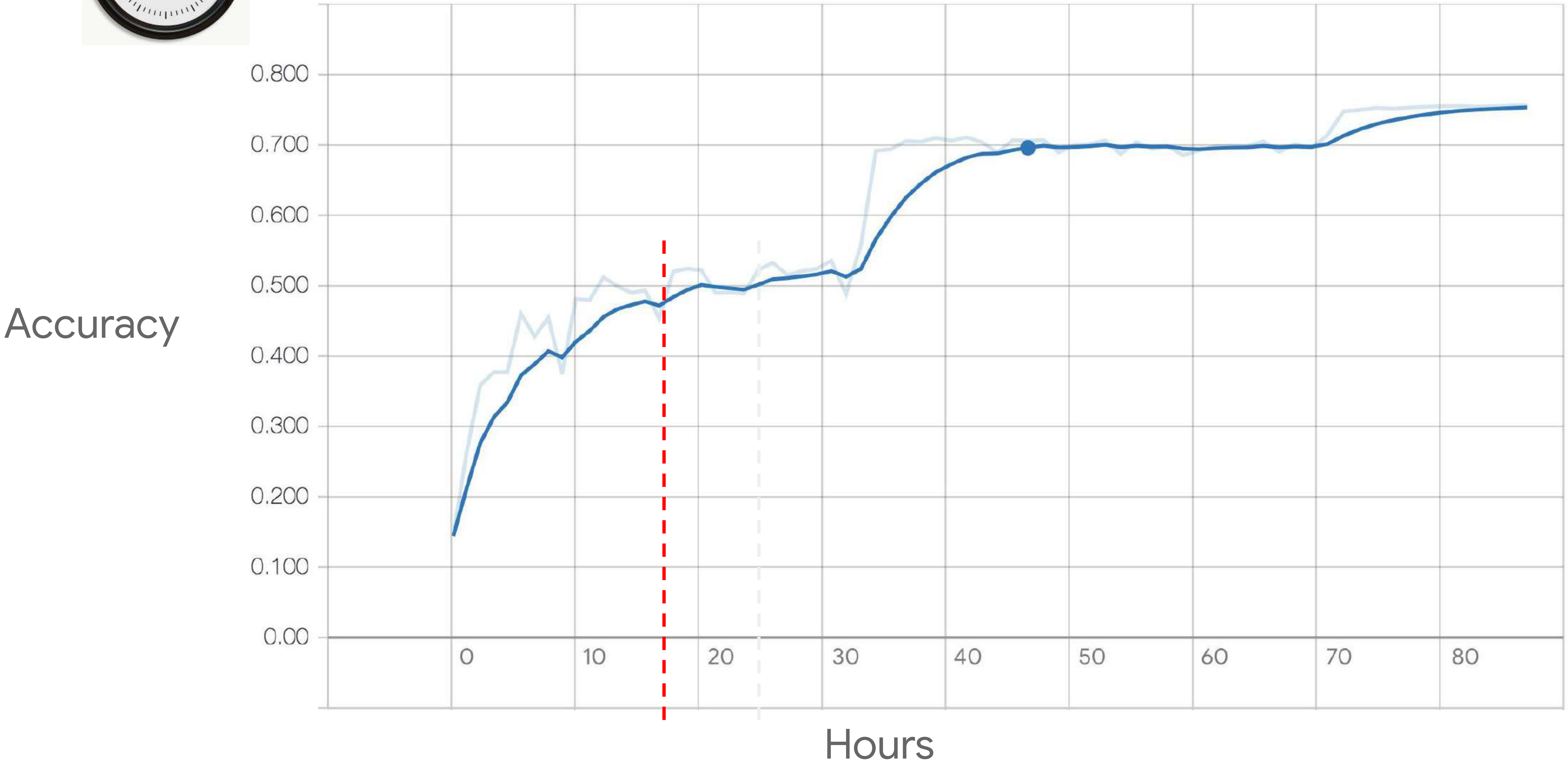
# Model Training can take a long time

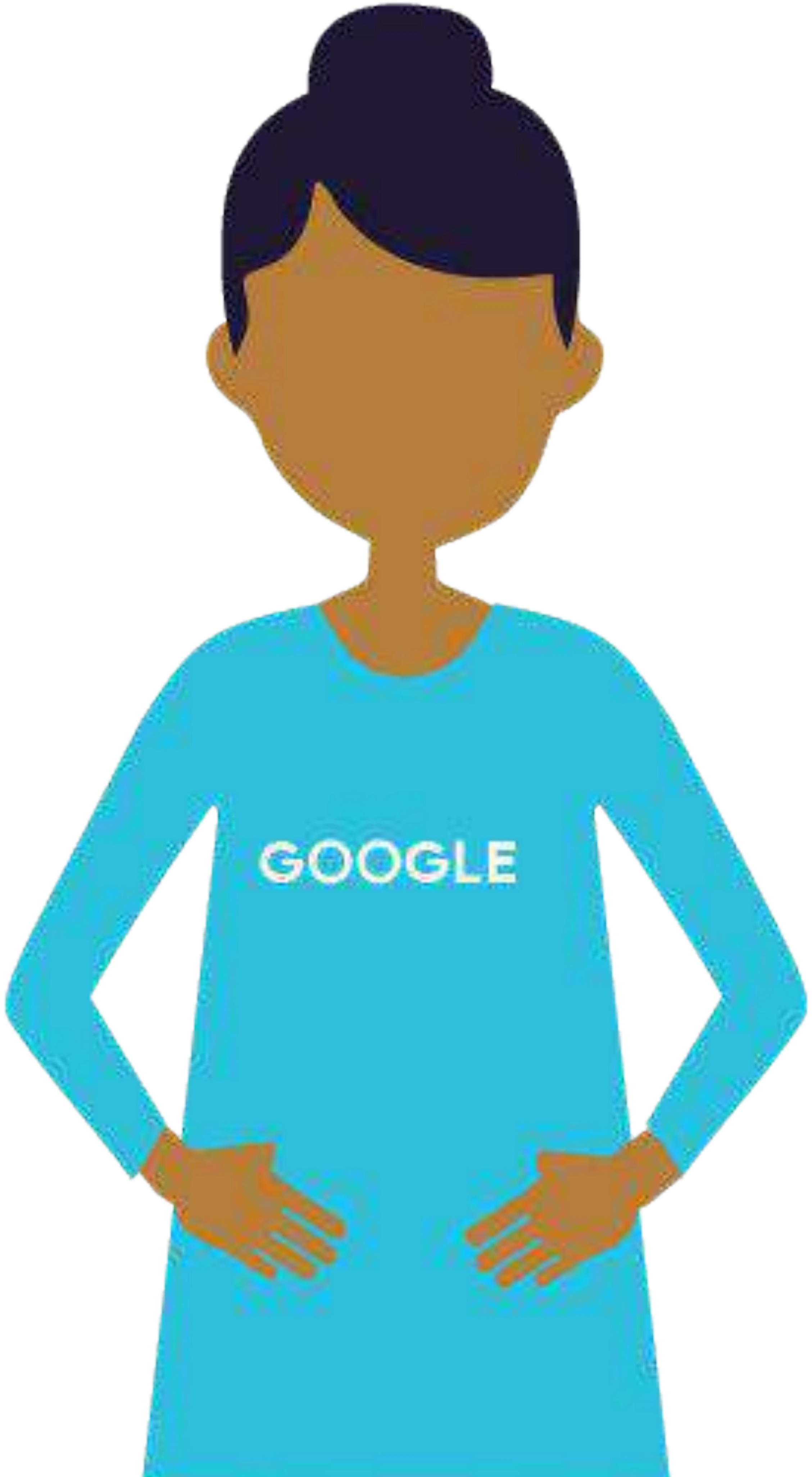


# Model Training can take a long time

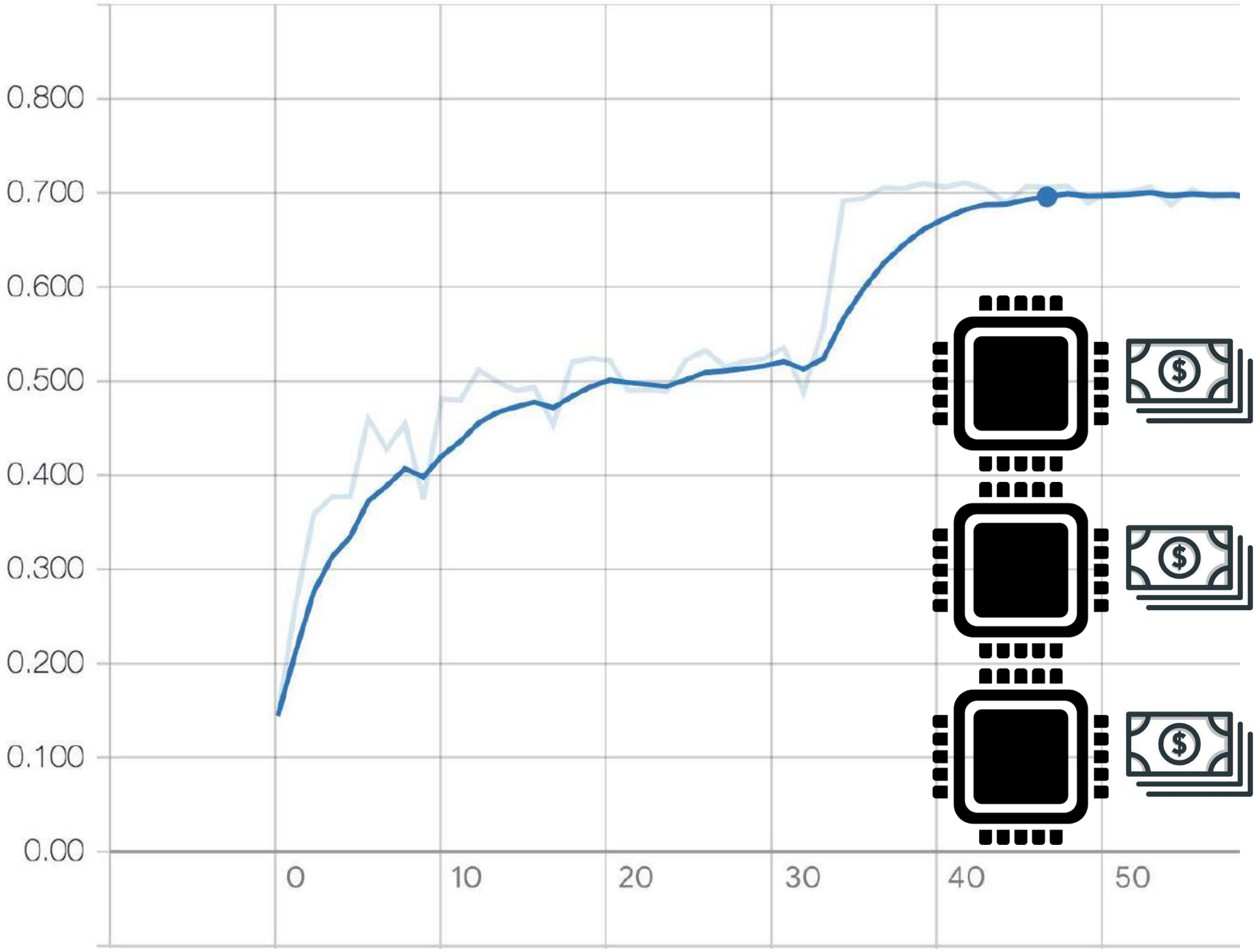


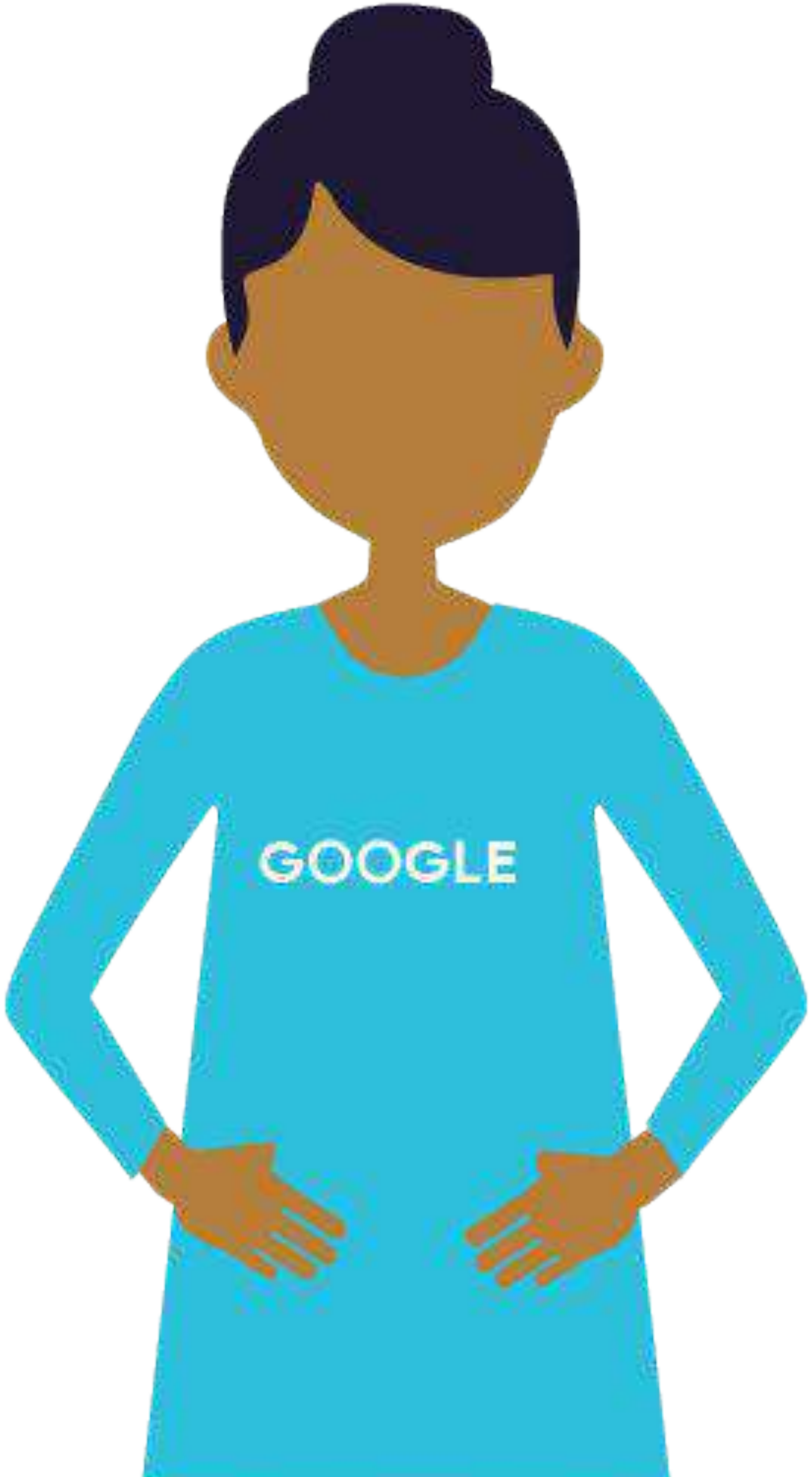
# Model Training can take a long time



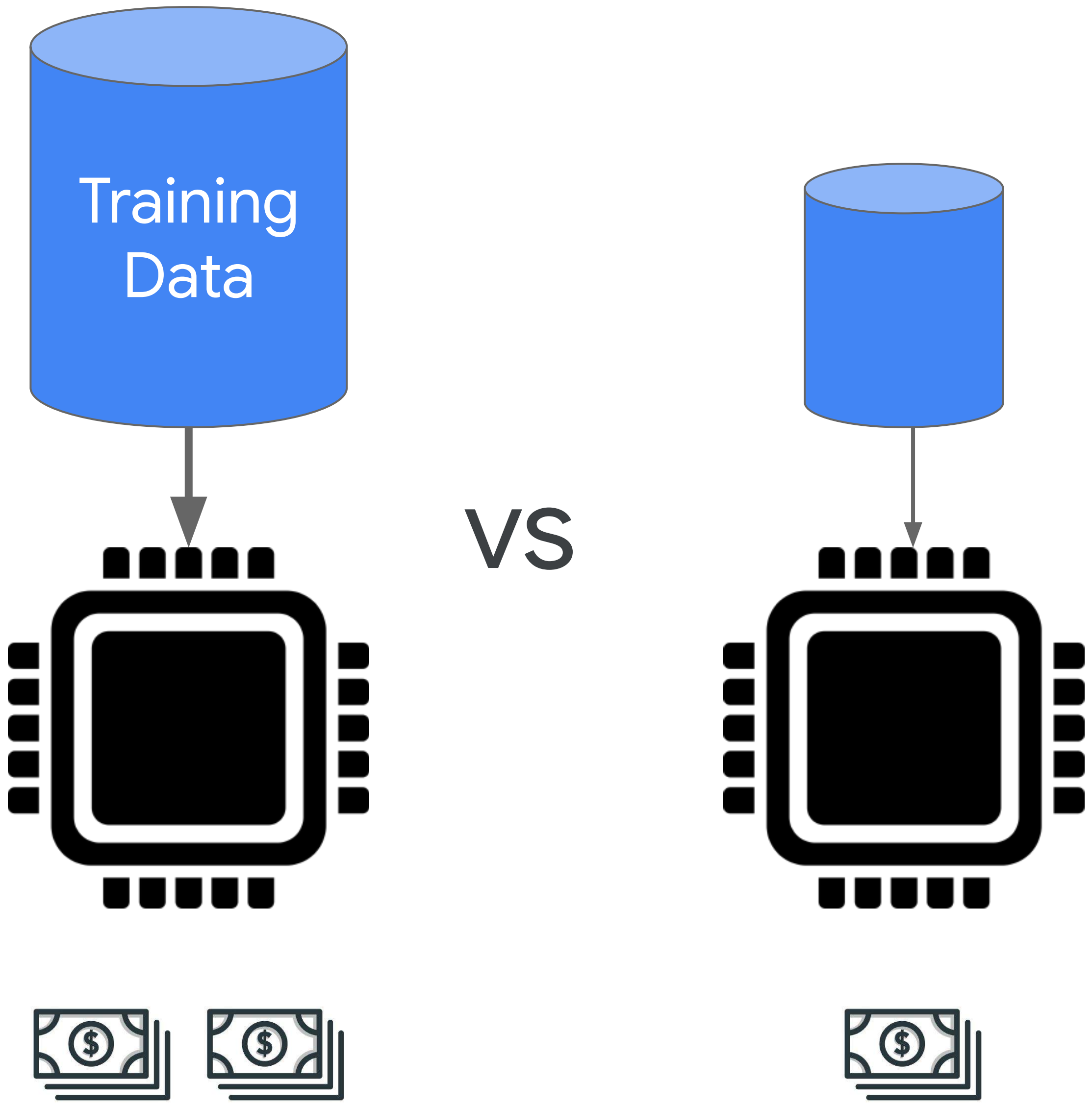


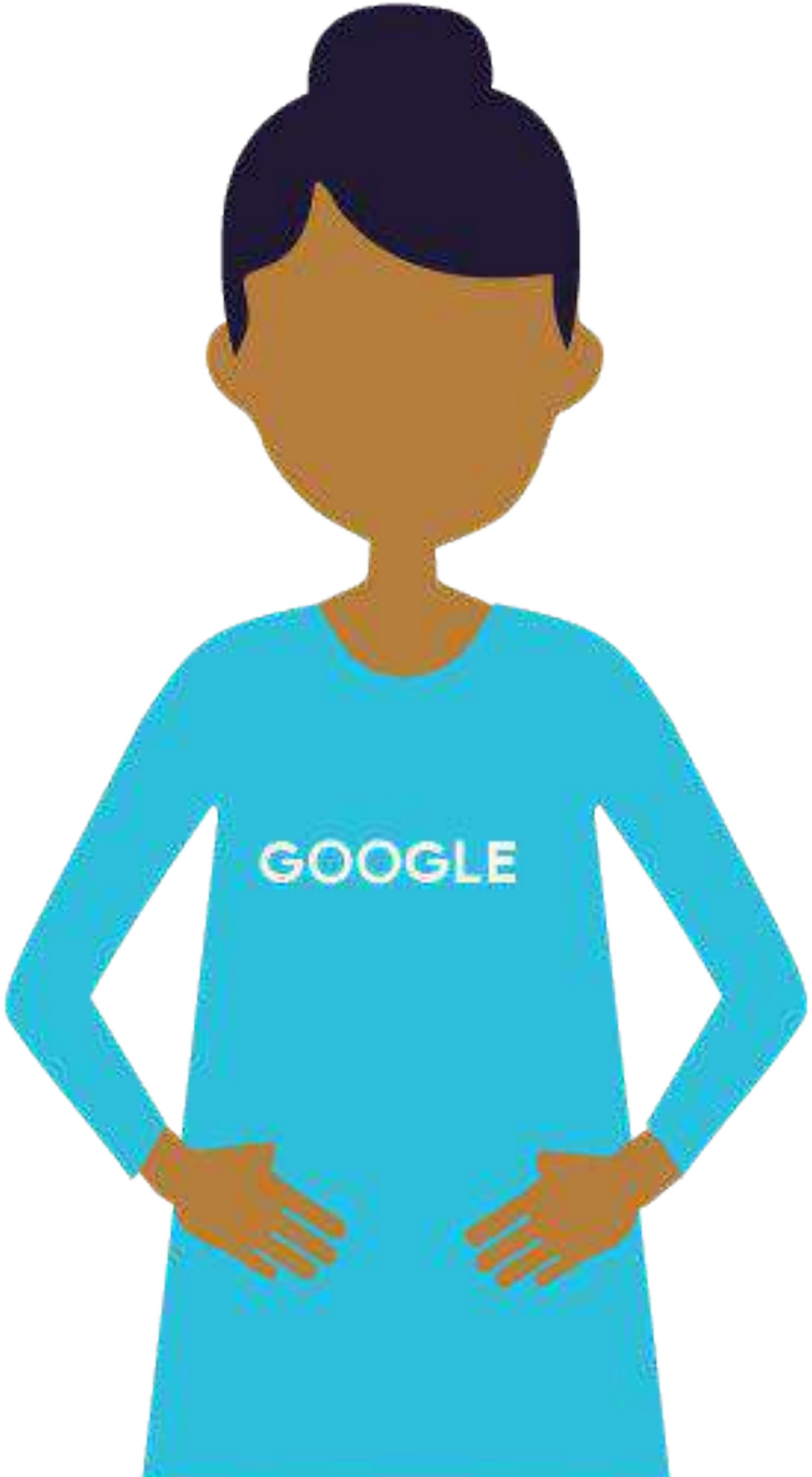

# Analyze Benefit of Model vs Running Cost



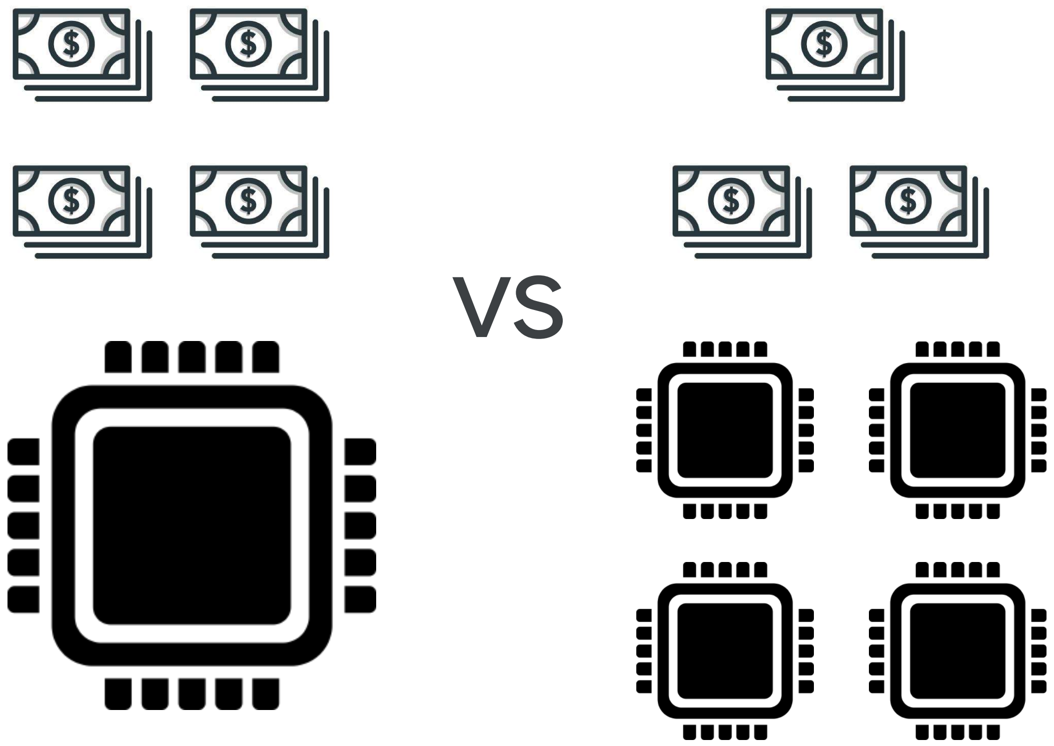


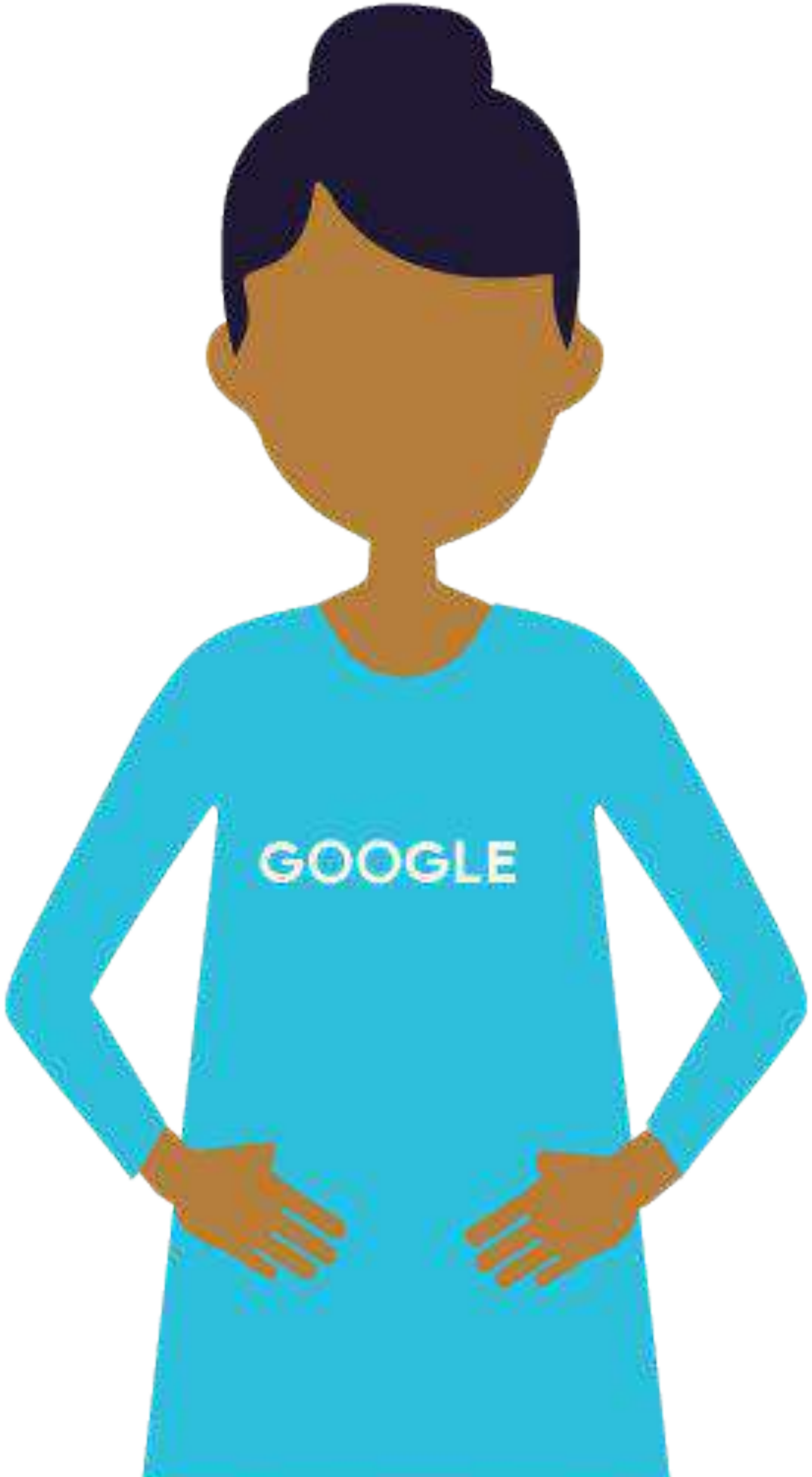

# Optimize training dataset size



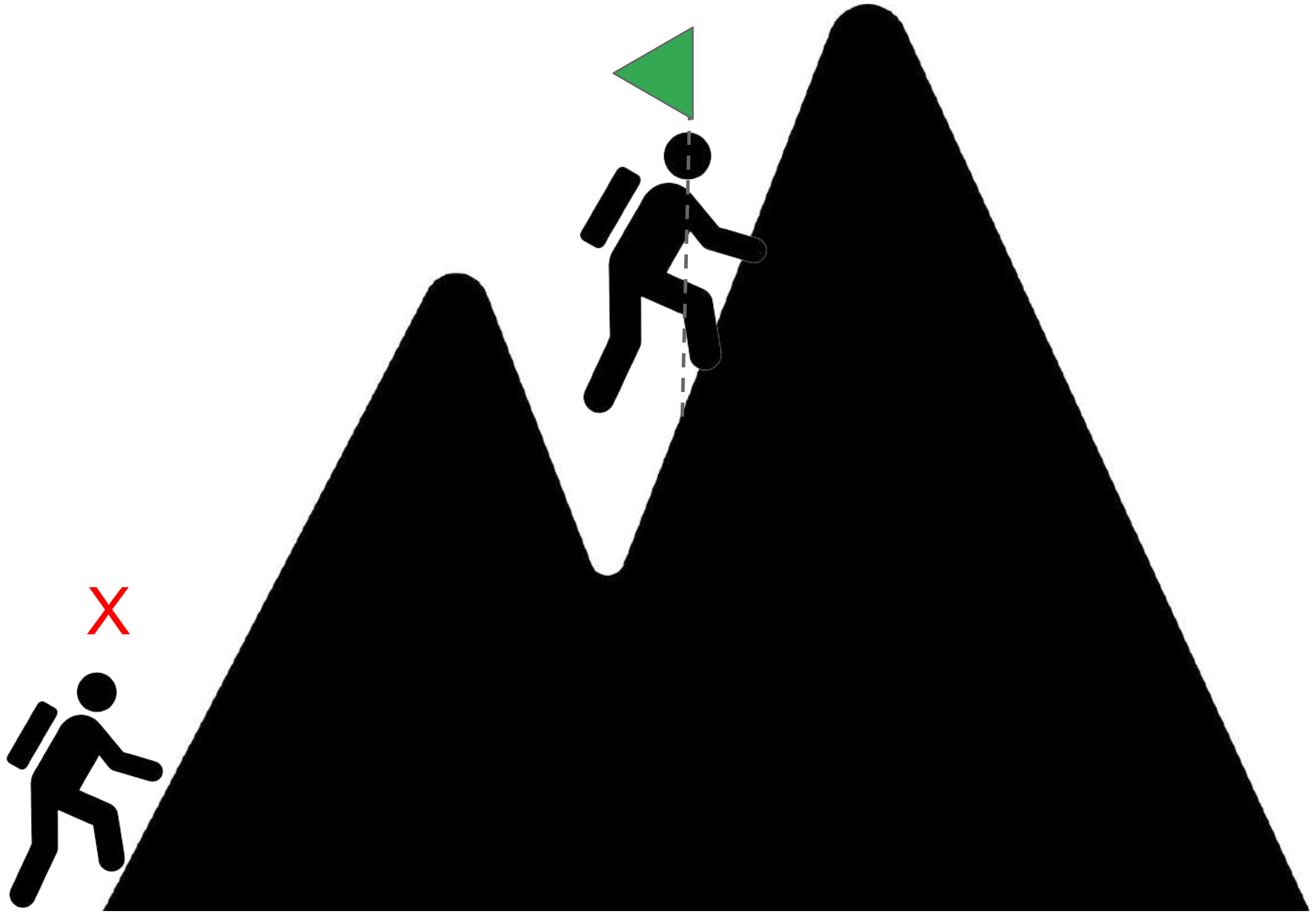



# Choosing optimized infrastructure






Use earlier model checkpoints



# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | |
|---|---|
| | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| **Constraint** | Input / Output |
| --- | --- |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| **Constraint** | Input / Output | CPU |
| --- | --- | --- |
| | | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
| --- | --- | --- | --- |
| | | | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
| --- | --- | --- | --- |
| **Commonly Occurs** | Large inputs<br>Input requires parsing<br>Small models | | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
|---|---|---|---|
| **Commonly Occurs** | Large inputs<br>Input requires parsing<br>Small models | Expensive computations<br>Underpowered Hardware | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
| --- | --- | --- | --- |
| **Commonly Occurs** | Large inputs<br>Input requires parsing<br>Small models | Expensive computations<br>Underpowered Hardware | Large number of inputs<br>Complex model |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
|---|---|---|---|
| Commonly Occurs | Large inputs<br>Input requires parsing<br>Small models | Expensive computations<br>Underpowered Hardware | Large number of inputs<br>Complex model |
| **Take Action** | Store efficiently<br>Parallelize reads<br>Consider batch size | | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
| --- | --- | --- | --- |
| Commonly Occurs | Large inputs<br>Input requires parsing<br>Small models | Expensive computations<br>Underpowered Hardware | Large number of inputs<br>Complex model |
| **Take Action** | Store efficiently<br>Parallelize reads<br>Consider batch size | Train on faster accel.<br>Upgrade processor<br>Run on TPUs<br>Simplify model | |

# Tuning Performance to reduce training time, reduce cost, and increase scale

| Constraint | Input / Output | CPU | Memory |
|---|---|---|---|
| Commonly Occurs | Large inputs<br>Input requires parsing<br>Small models | Expensive computations<br>Underpowered Hardware | Large number of inputs<br>Complex model |
| **Take Action** | Store efficiently<br>Parallelize reads<br>Consider batch size | Train on faster accel.<br>Upgrade processor<br>Run on TPUs<br>Simplify model | Add more memory<br>Use fewer layers<br>Reduce batch size |

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

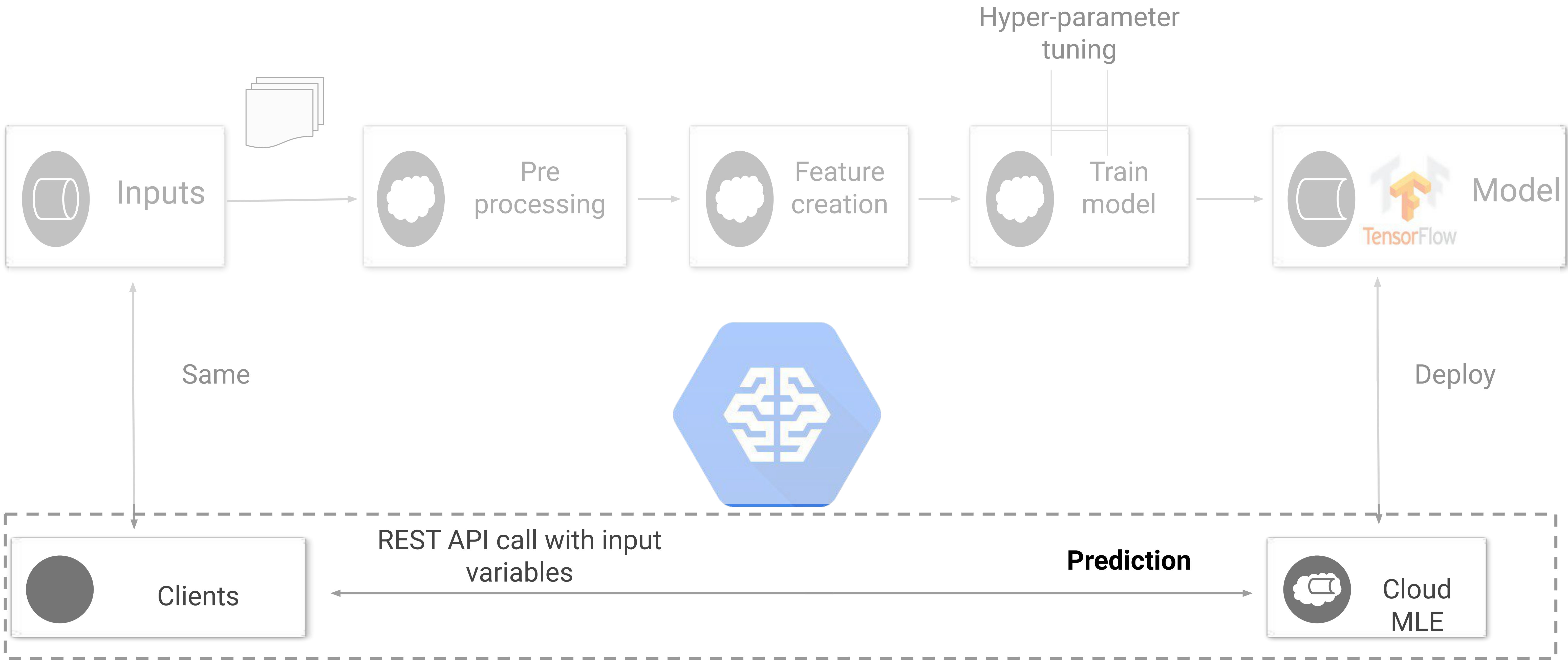
Lesson Title: **Aspects of Performance: Predictions**

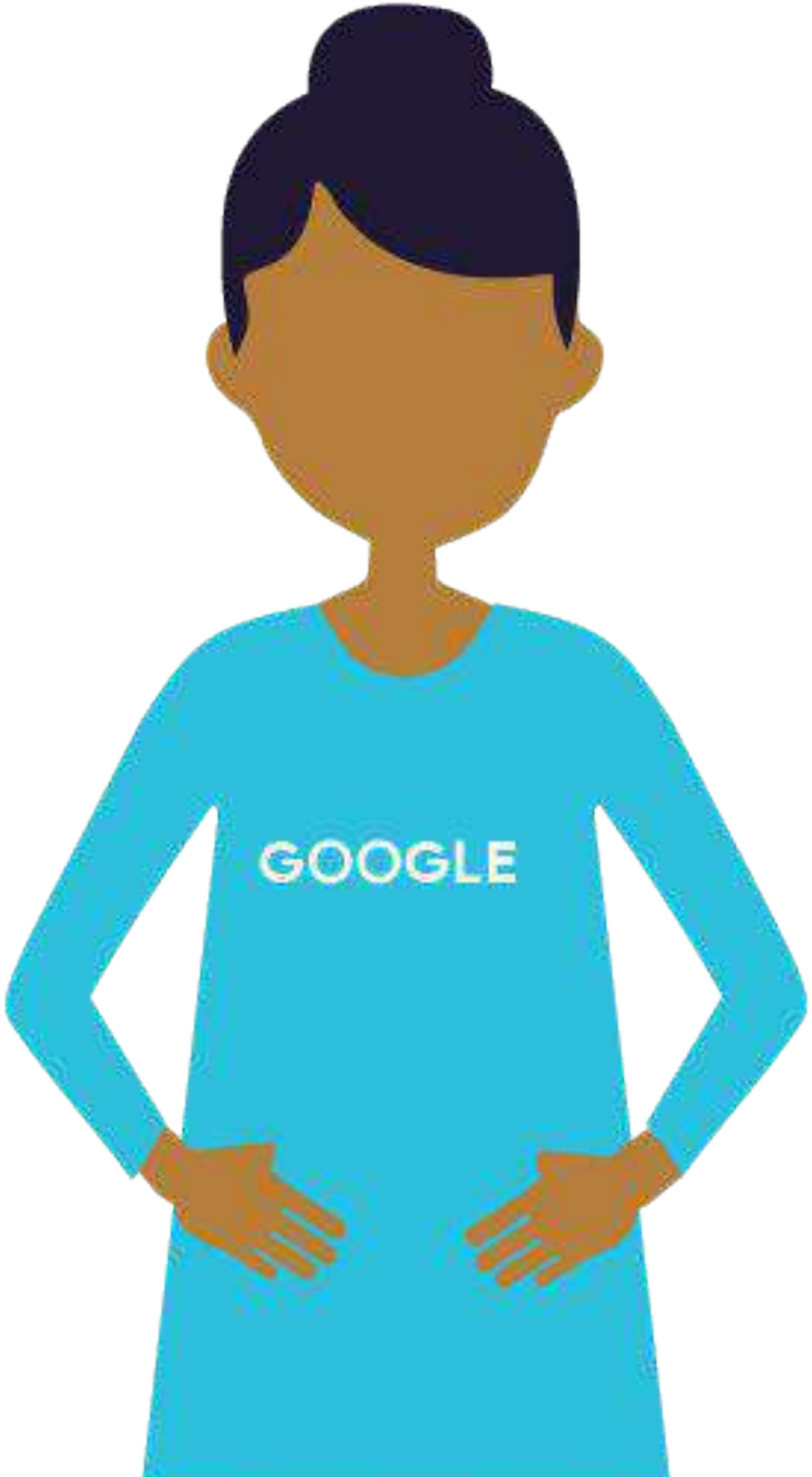
Format: Presenter

Presenter: Laurence Moroney

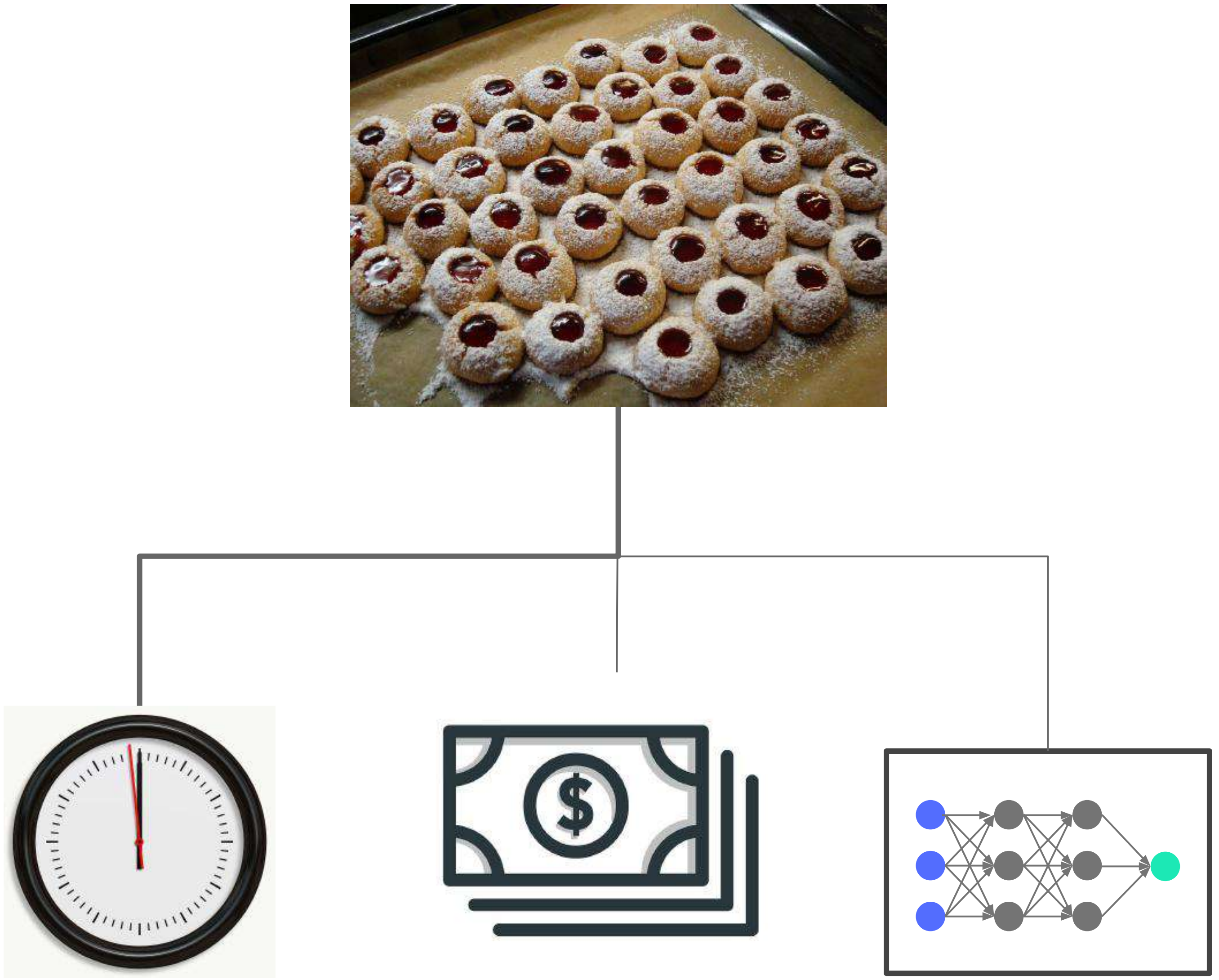
Video Name: T-PSML-O_4_l3_aspects_of_performance:_predictions

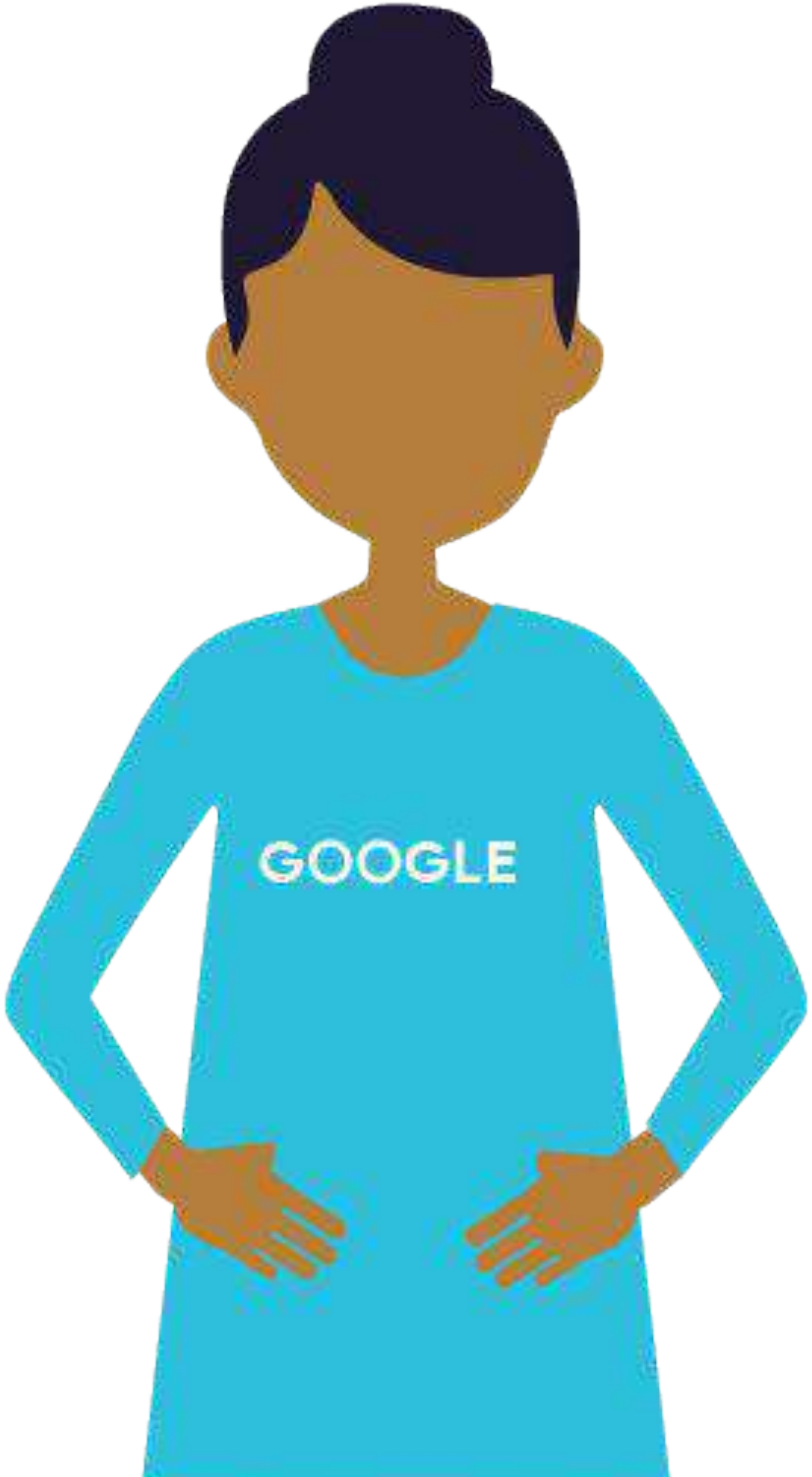
# Performance must consider prediction-time, not just training



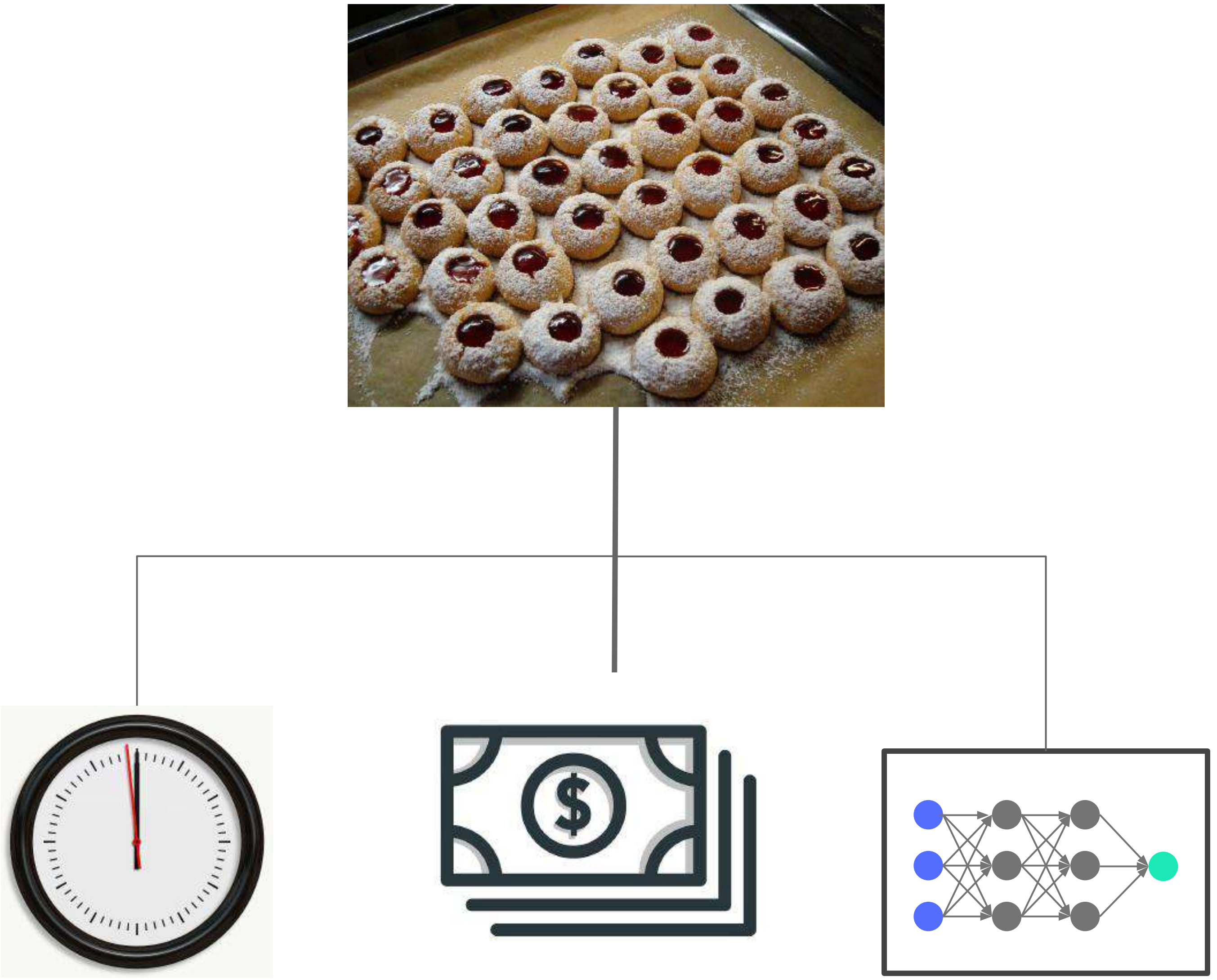


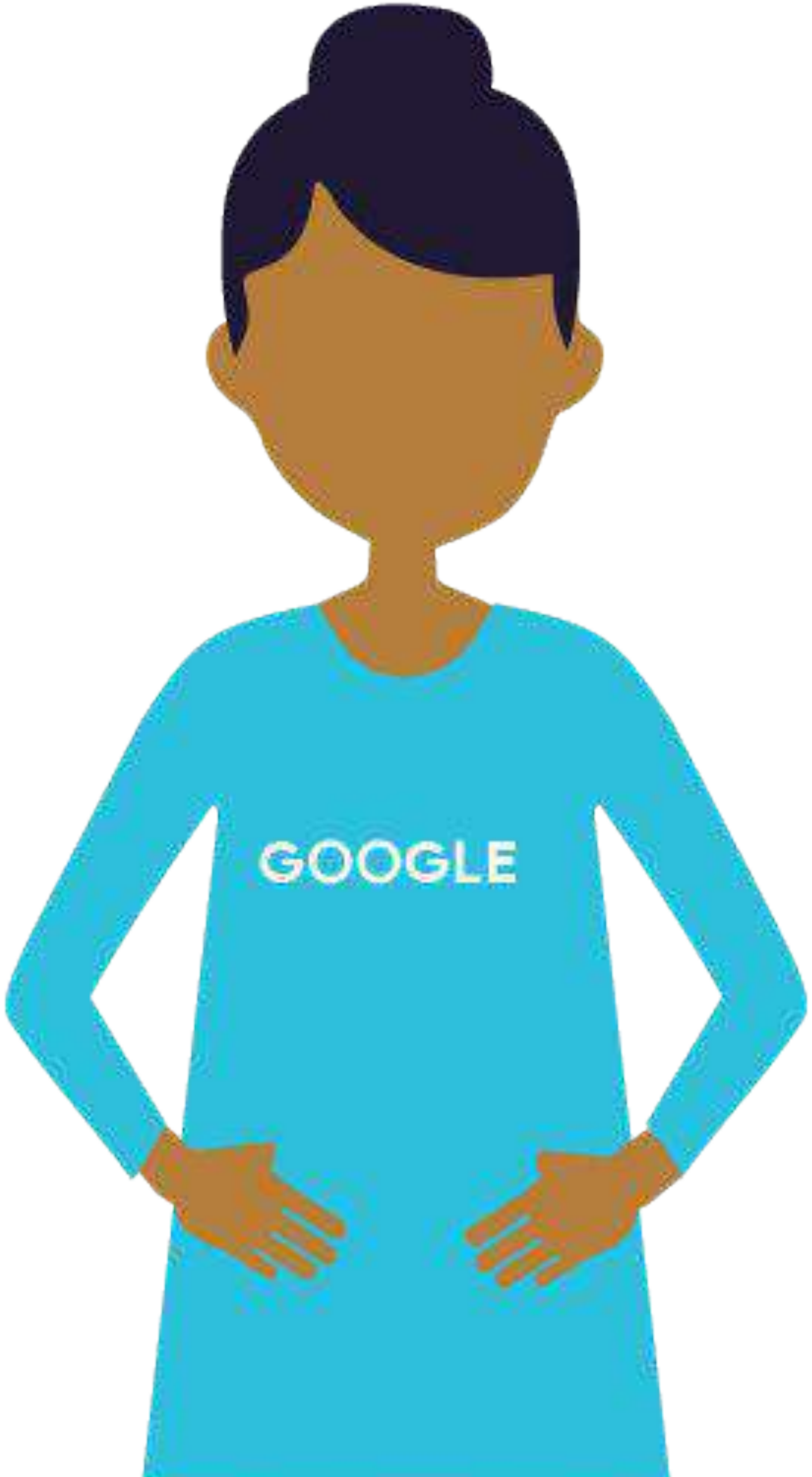

# Optimizing your Batch Prediction



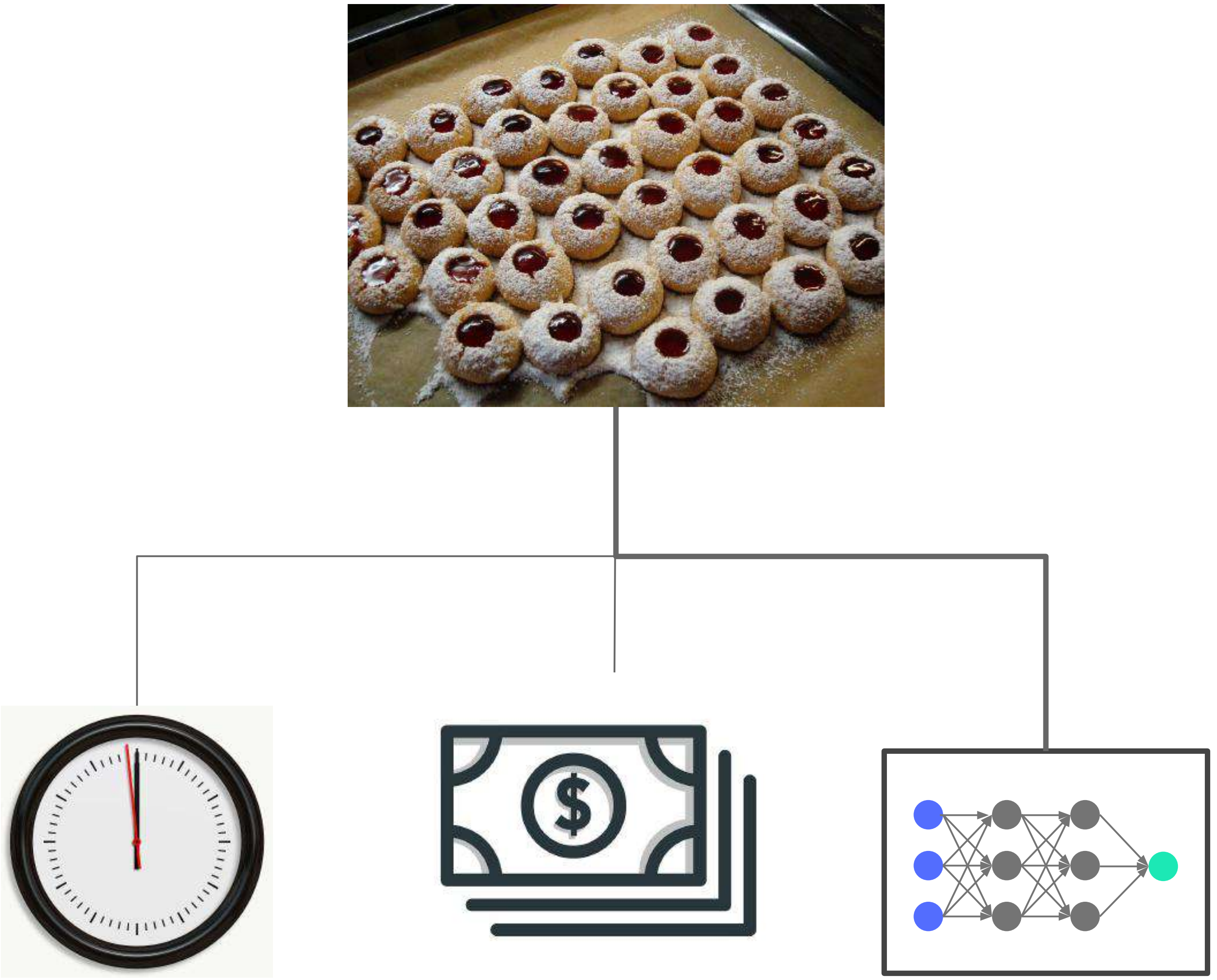


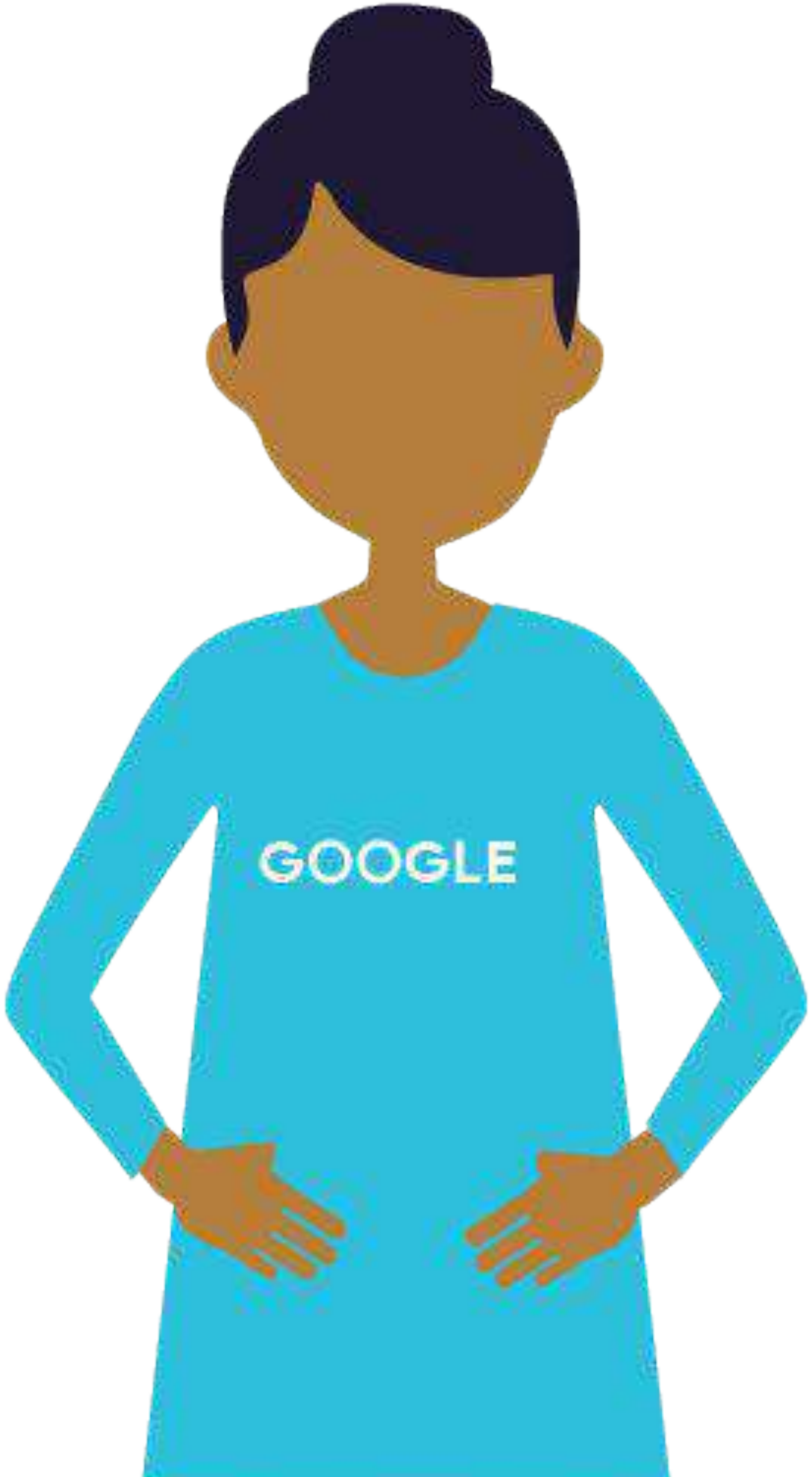

Optimizing your
Batch Prediction



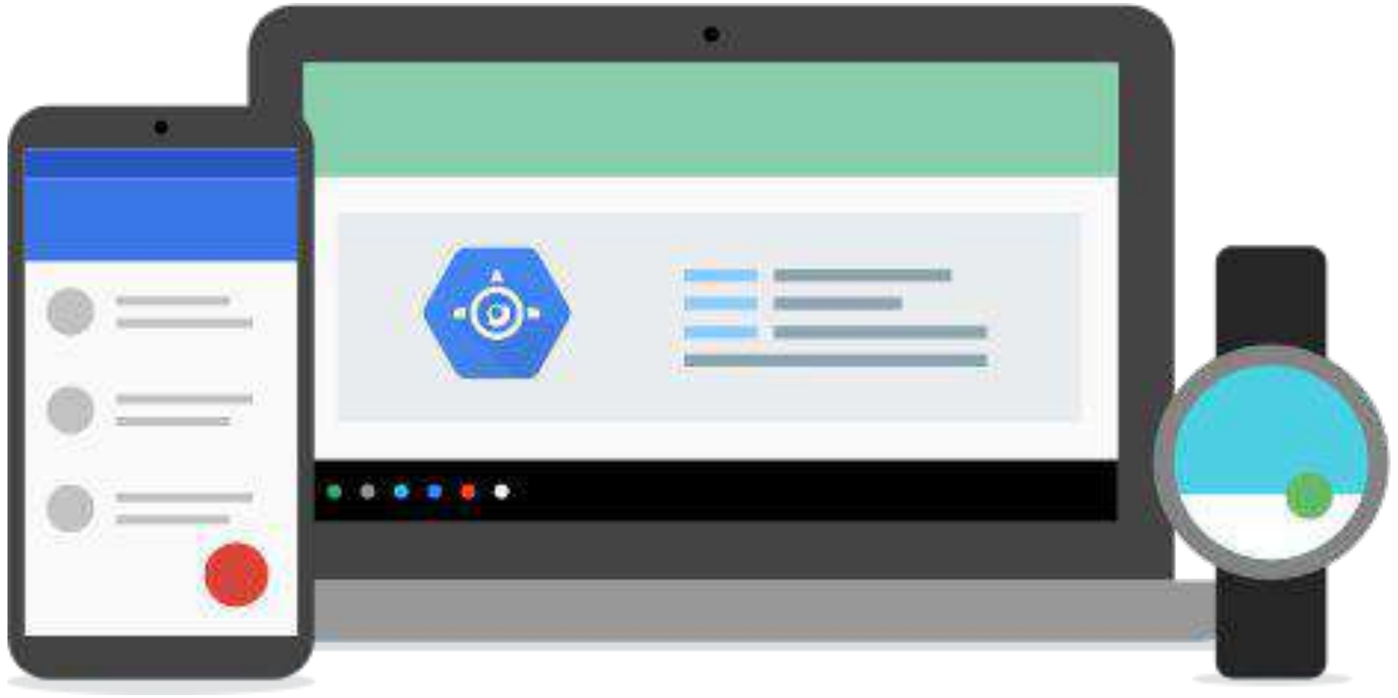


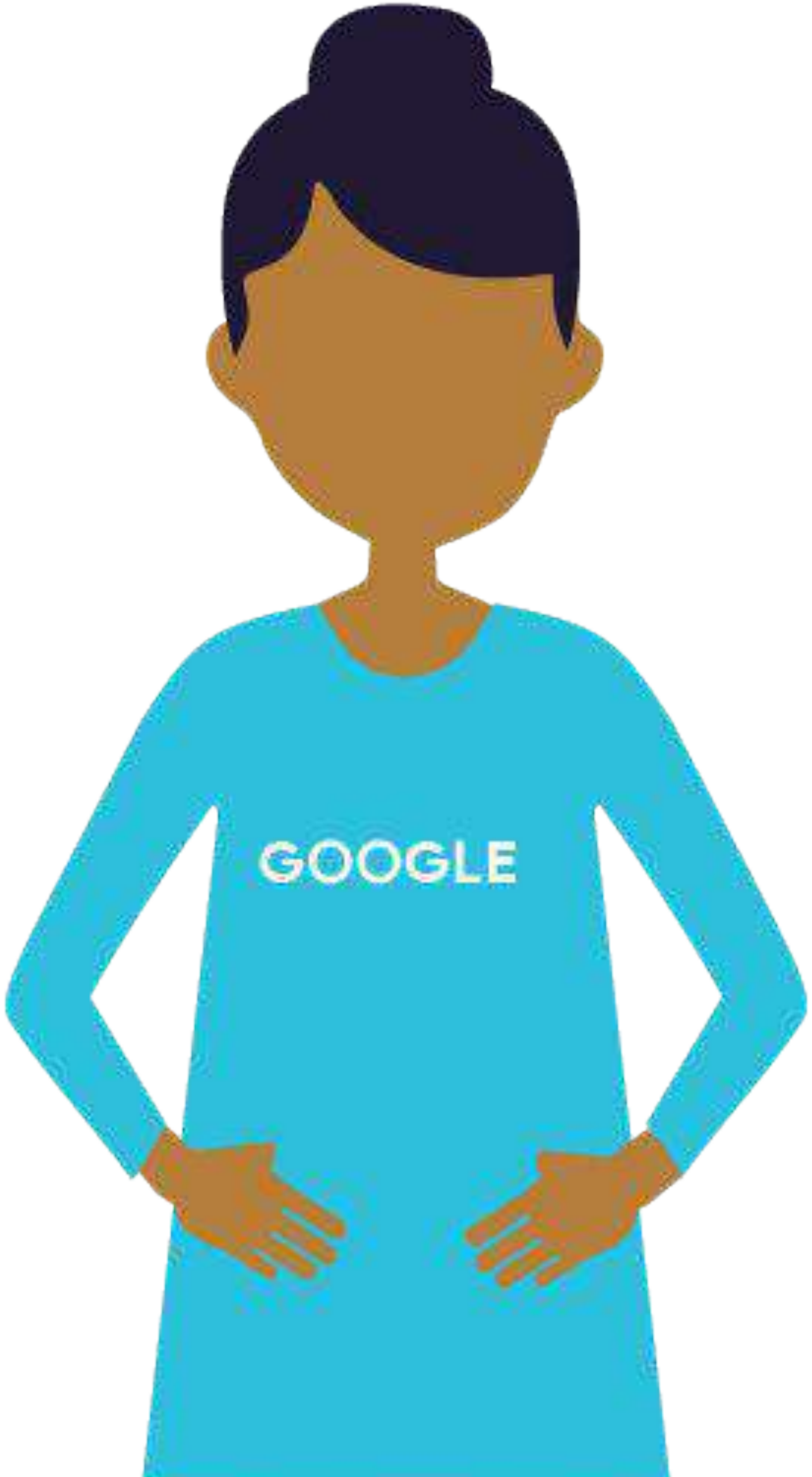

# Optimizing your Batch Prediction



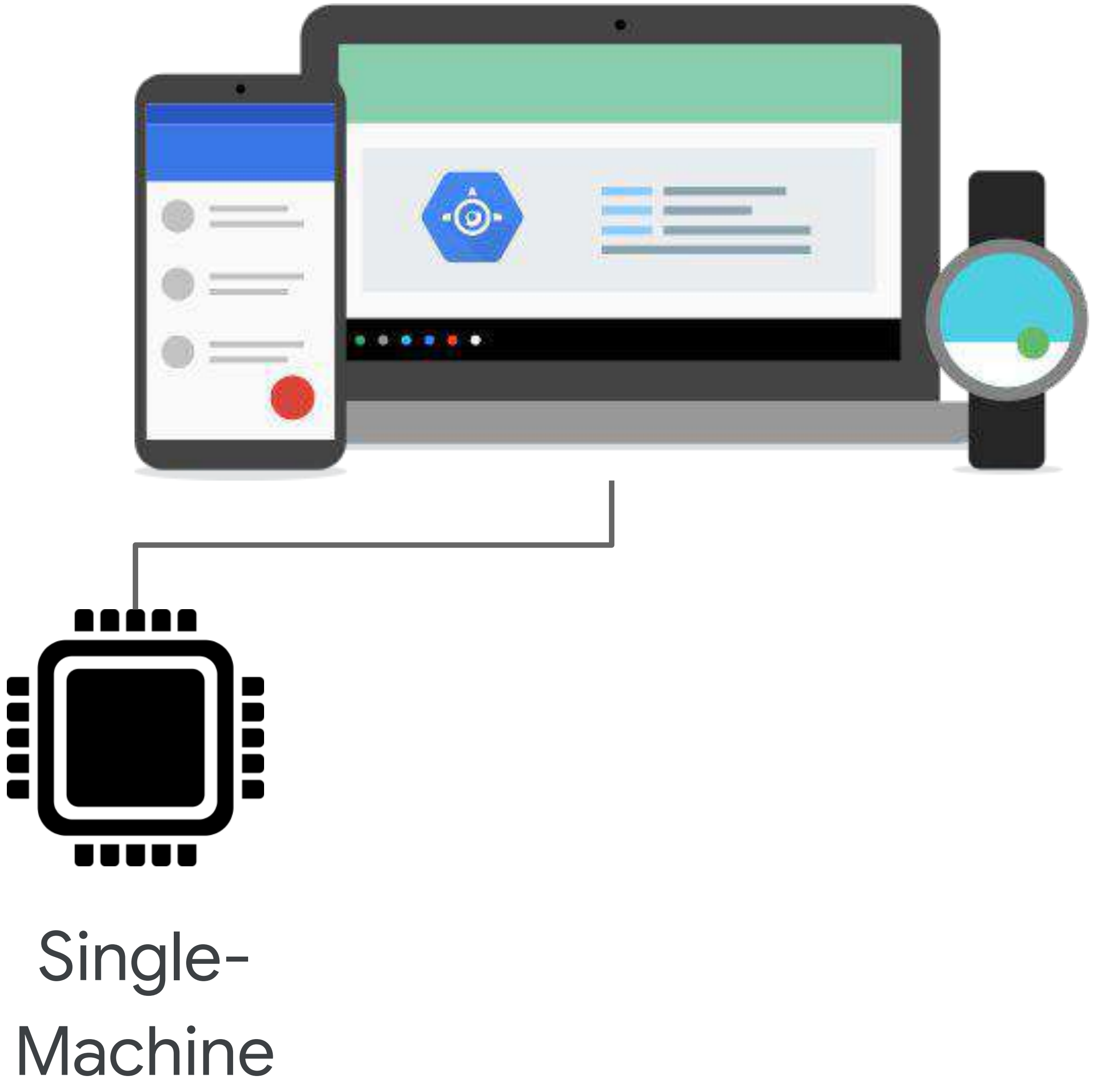


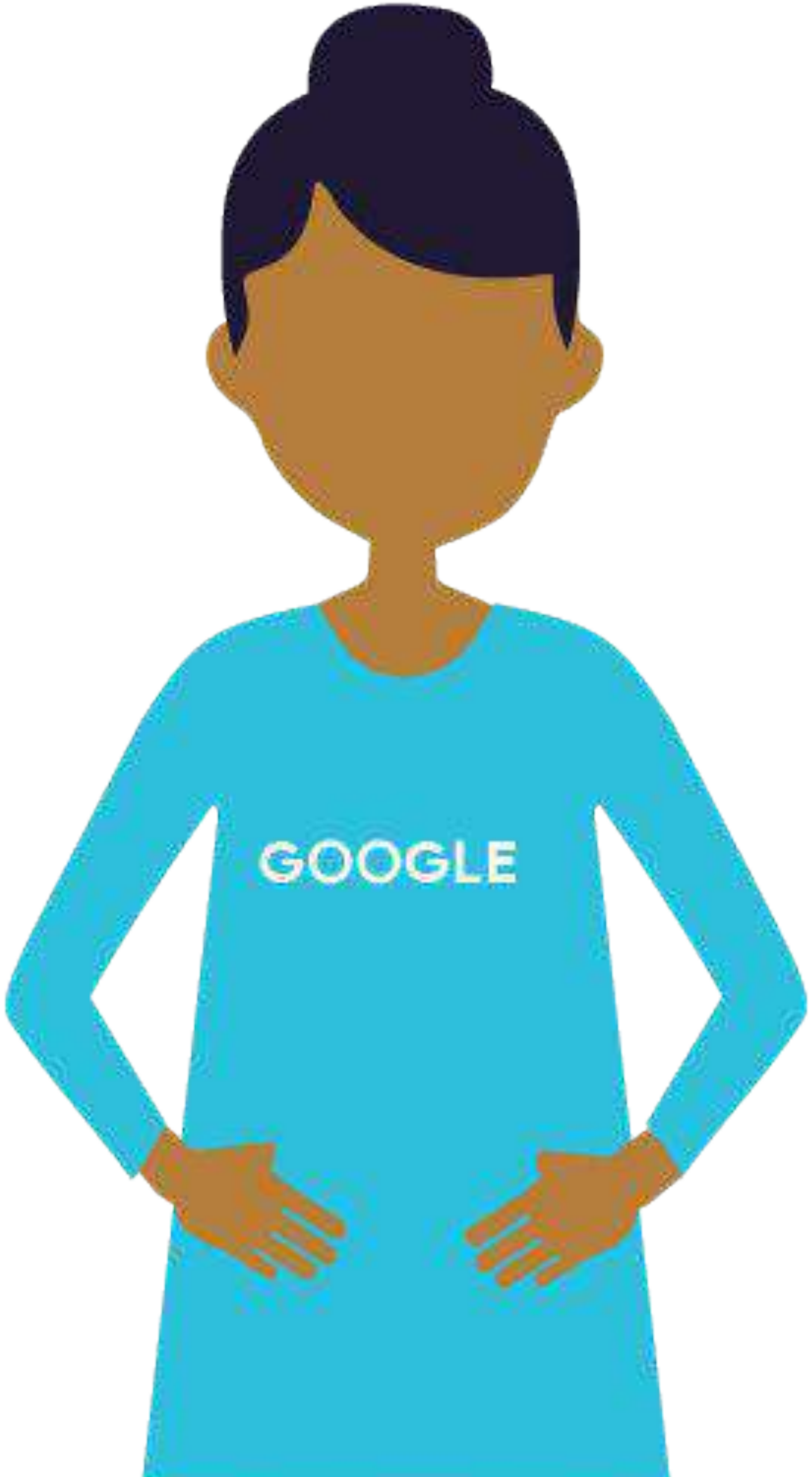

# Optimizing your Online Predictions



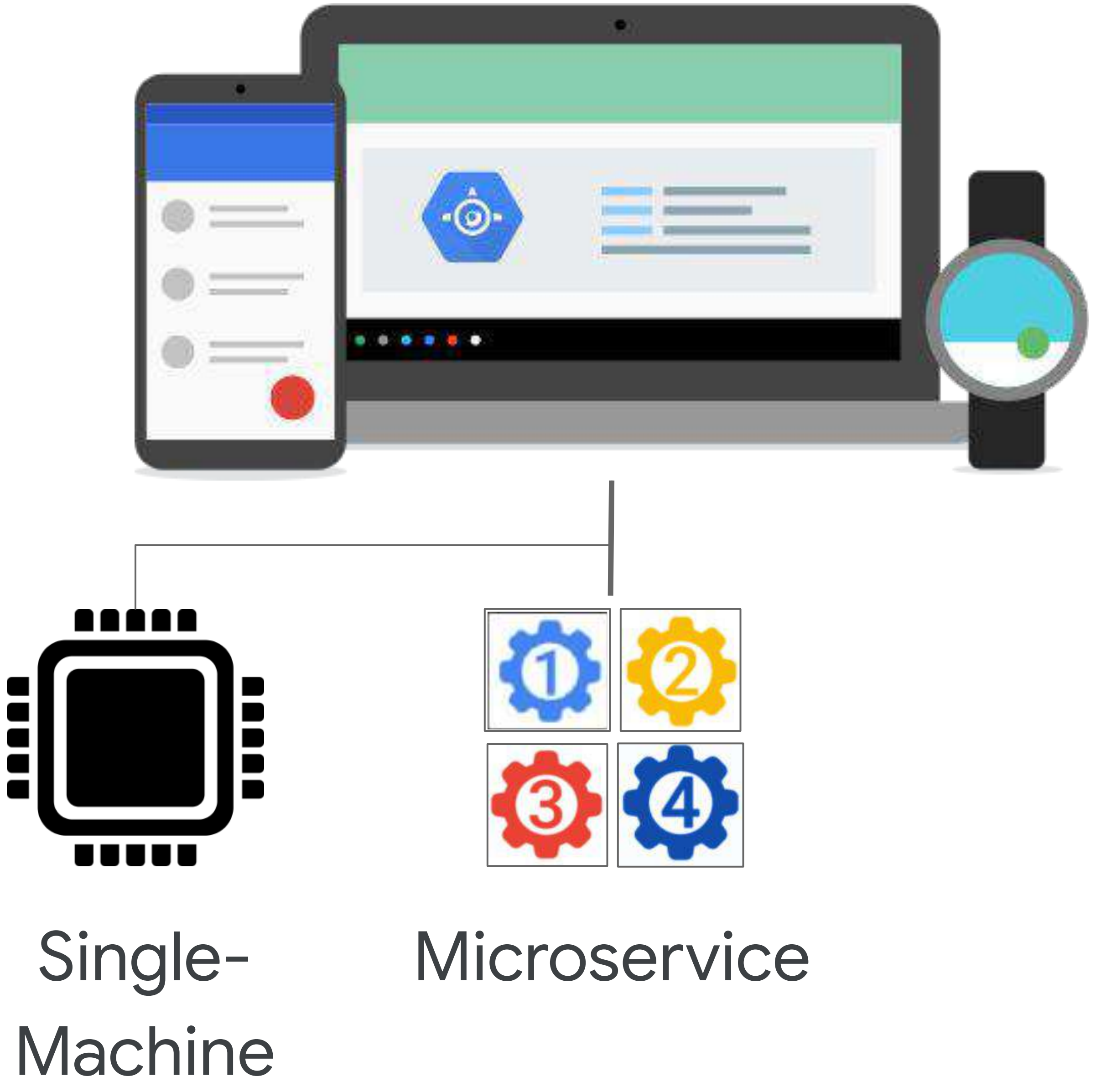



# Optimizing your Online Predictions

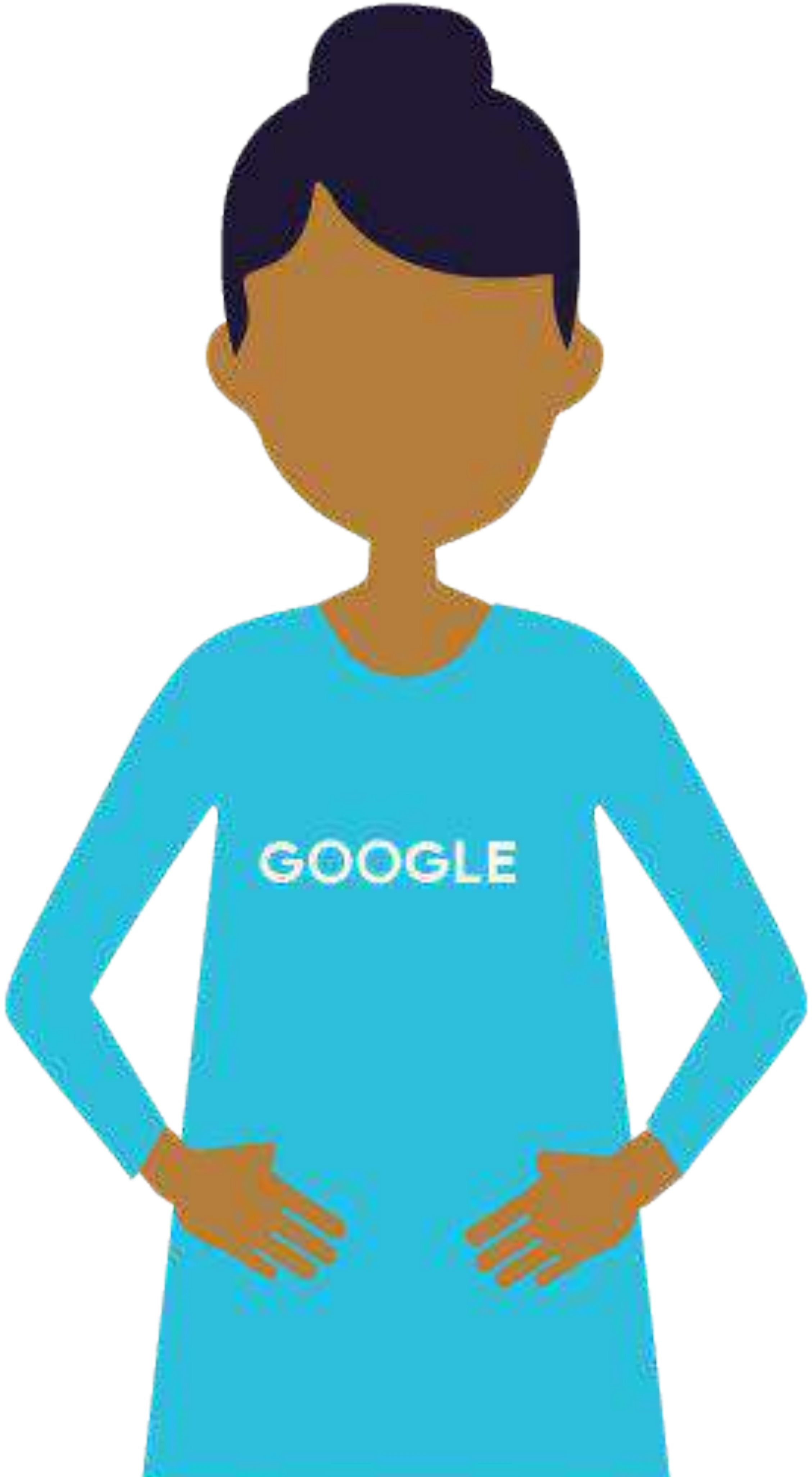


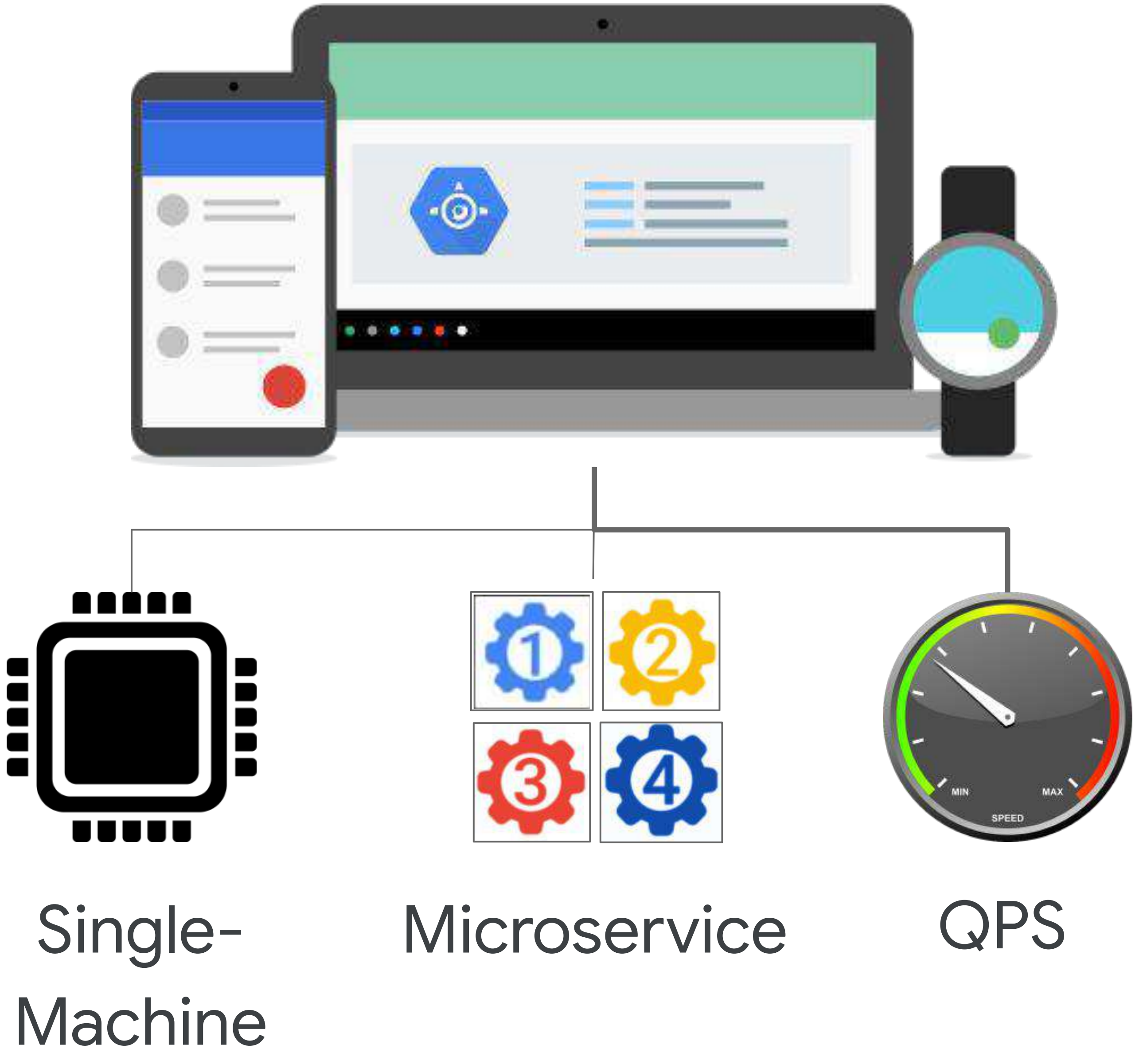


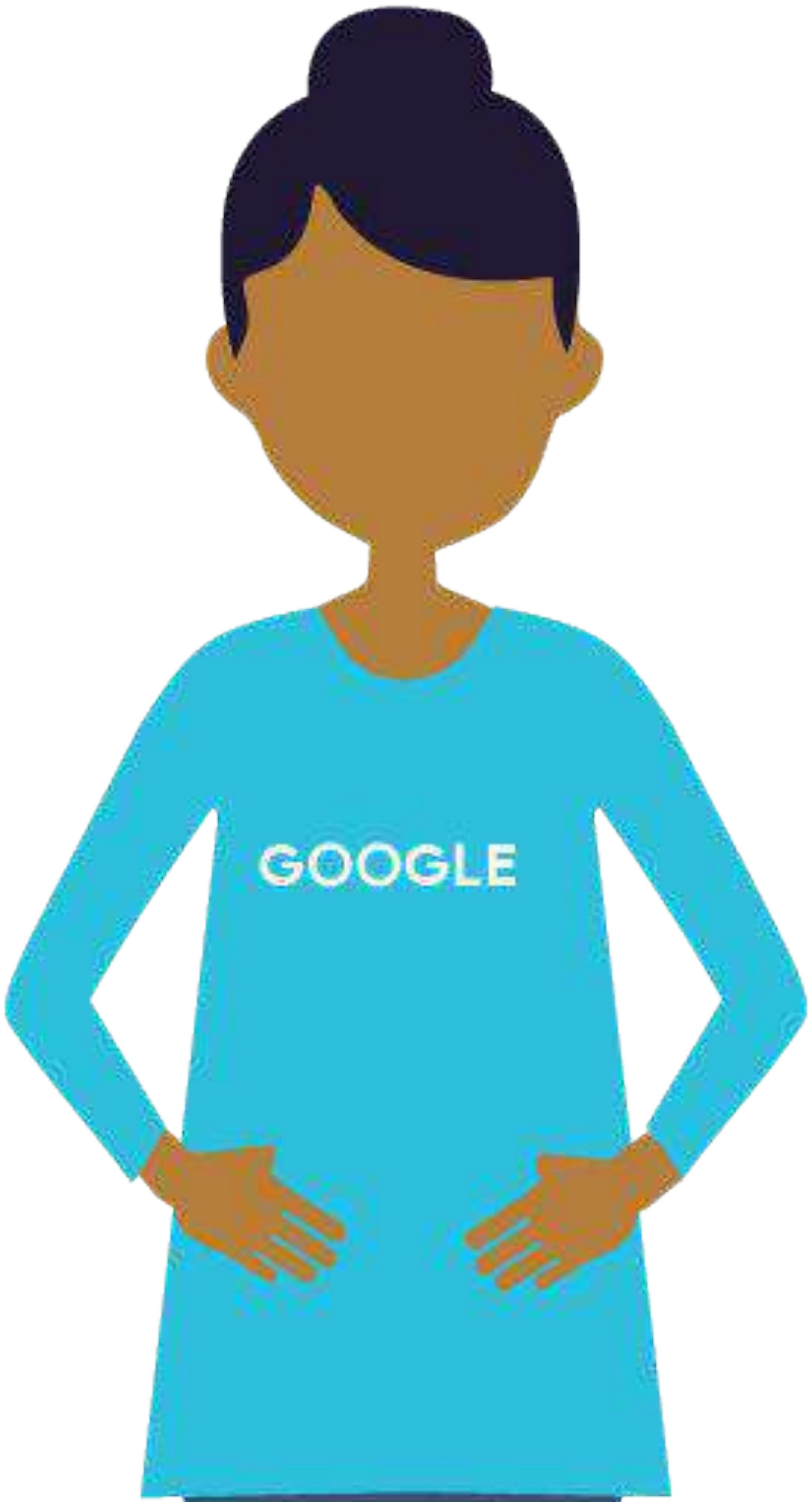

# Optimizing your Online Predictions



# Optimizing your Online Predictions






GOOGLE

Courses 7 - Production ML Systems

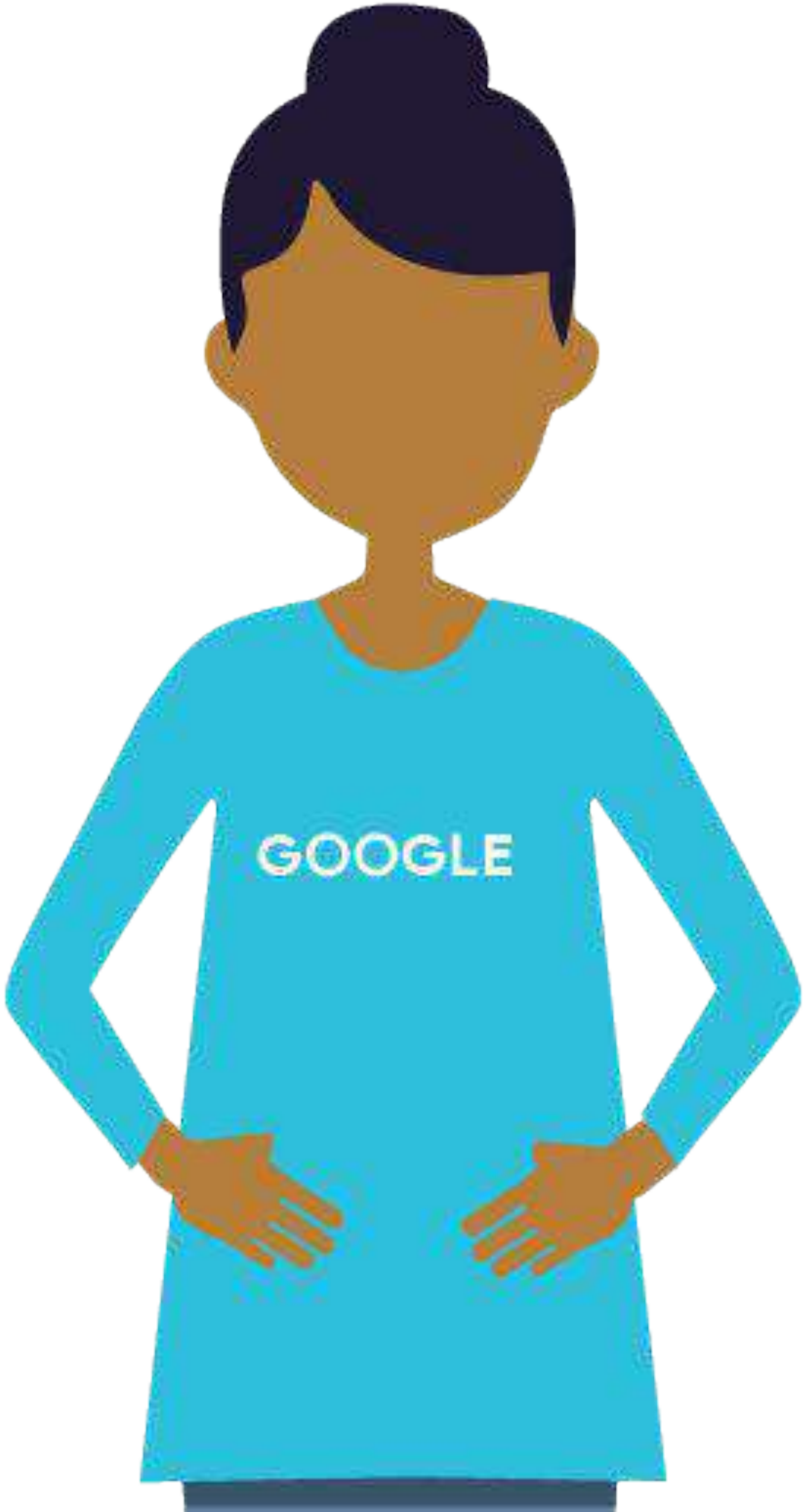
Module 4: Designing High-Performance ML Systems

Lesson Title: **Why distributed training?**

Format: Presenter

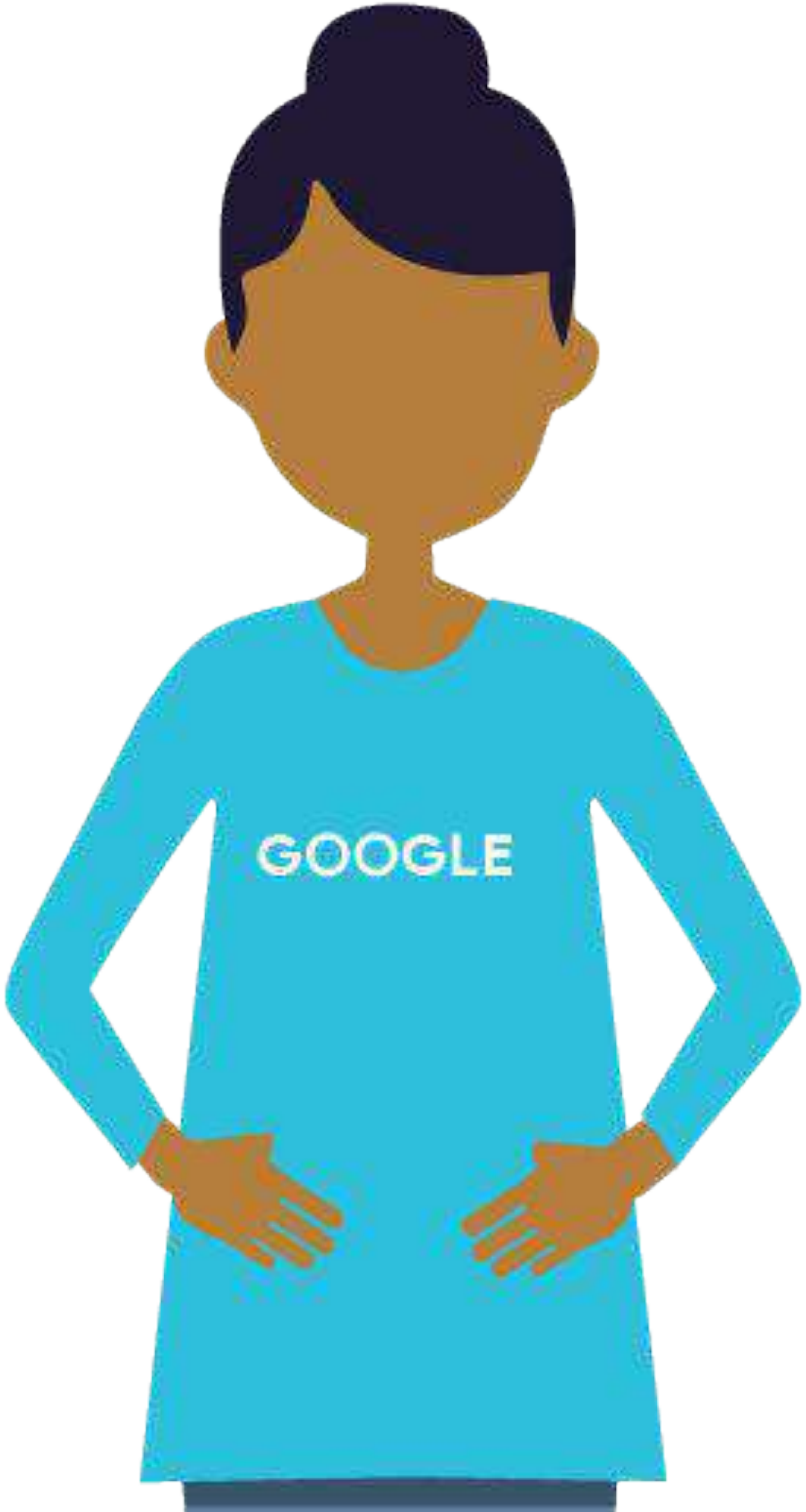
Presenter: Laurence Moroney

Video Name: T-PSML-O_4_I4_why_distributed_training

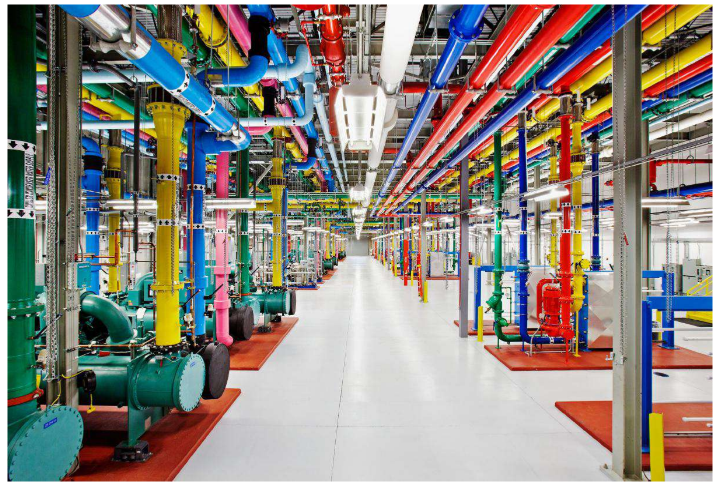



# Improving performance also adds complexity

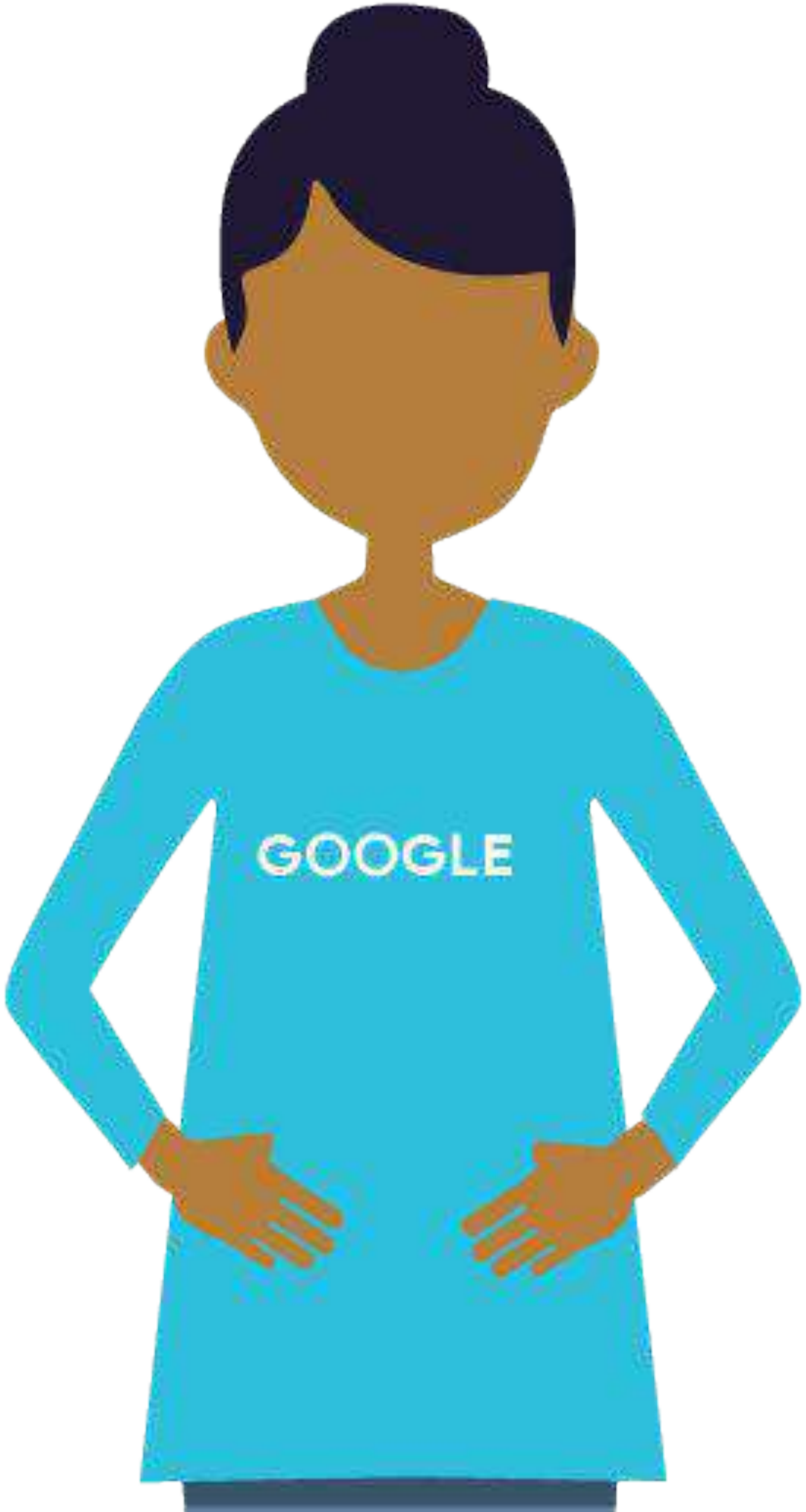
HETEROGENOUS SYSTEMS




Improving performance also adds complexity

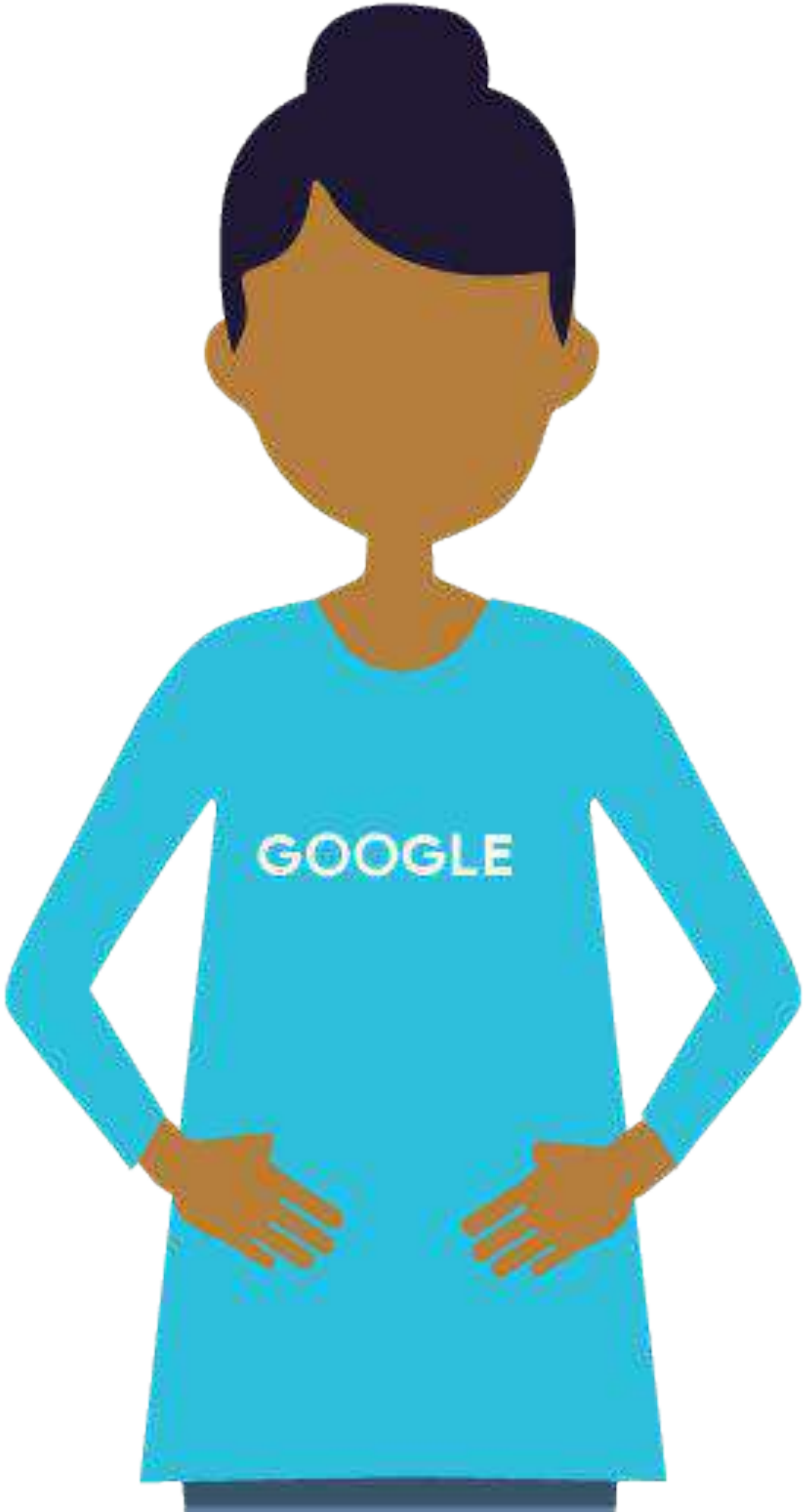


DISTRIBUTED SYSTEMS



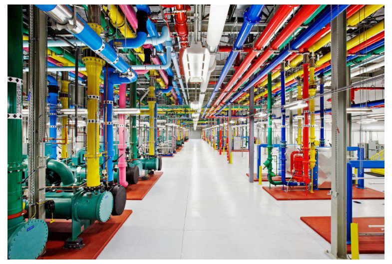

# Improving performance also adds complexity

MODEL ARCHITECTURES

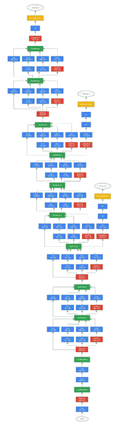



# Machine learning gets complex quickly

HETEROGENOUS SYSTEMS



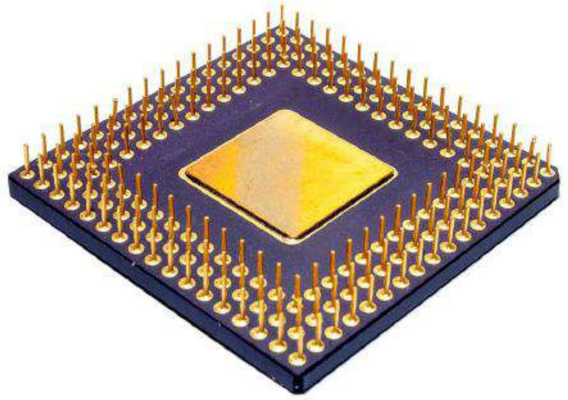
DISTRIBUTED SYSTEMS



MODEL ARCHITECTURES

# Heterogeneous systems require our code to work anywhere

CPU

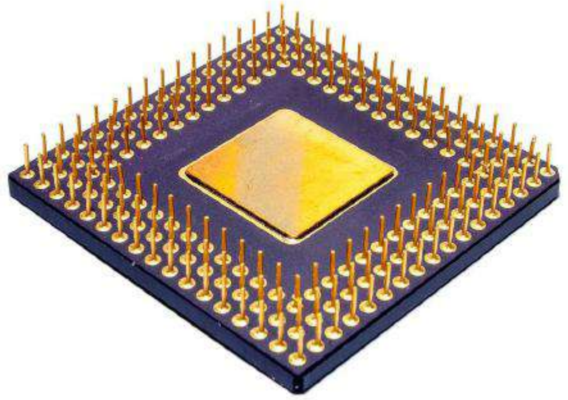


GPU

TPU

# Heterogeneous systems require our code to work anywhere

CPU

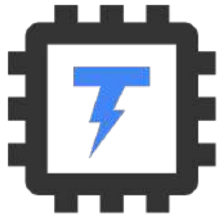


GPU

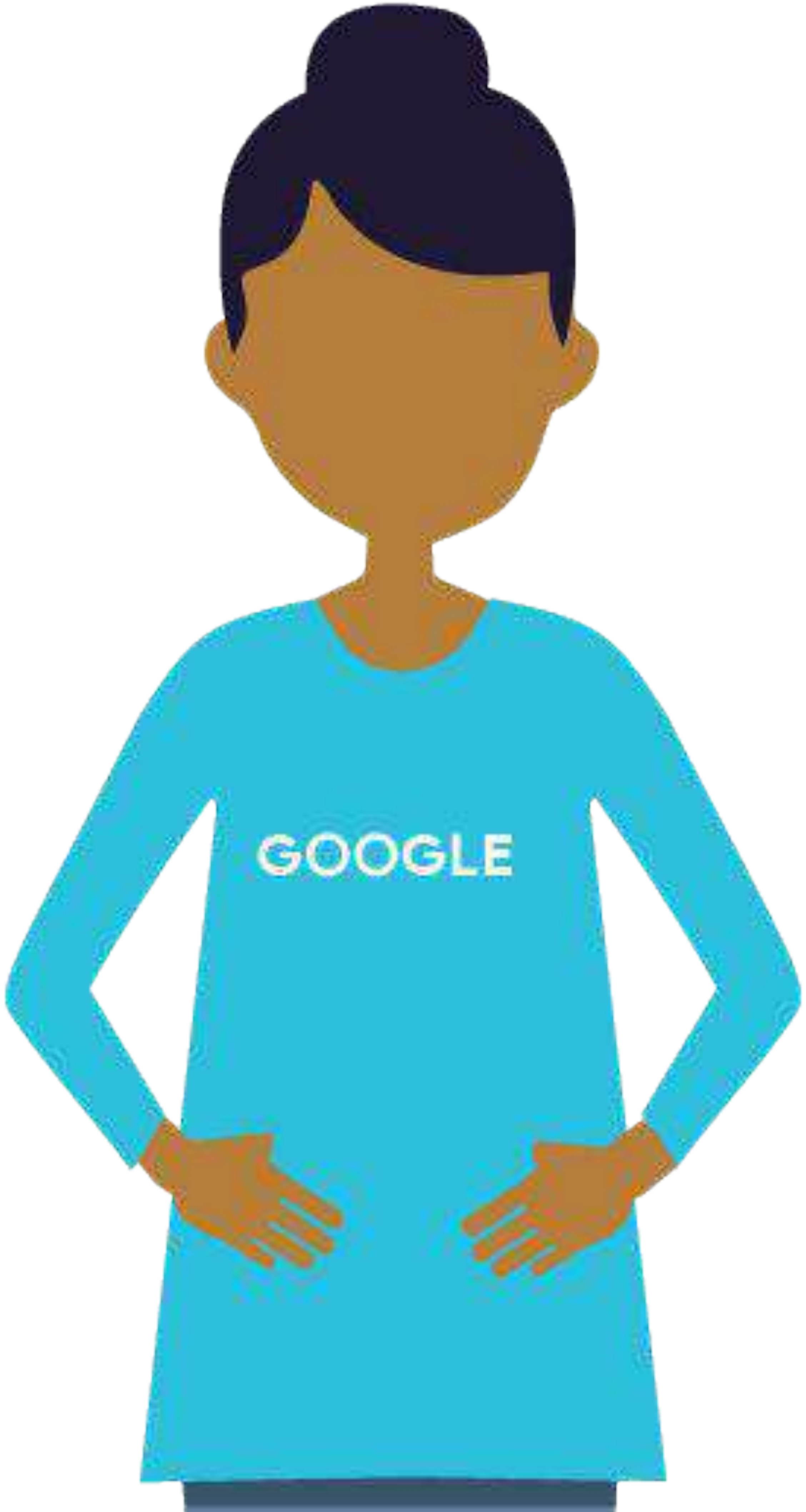
TPU

Android / iOS

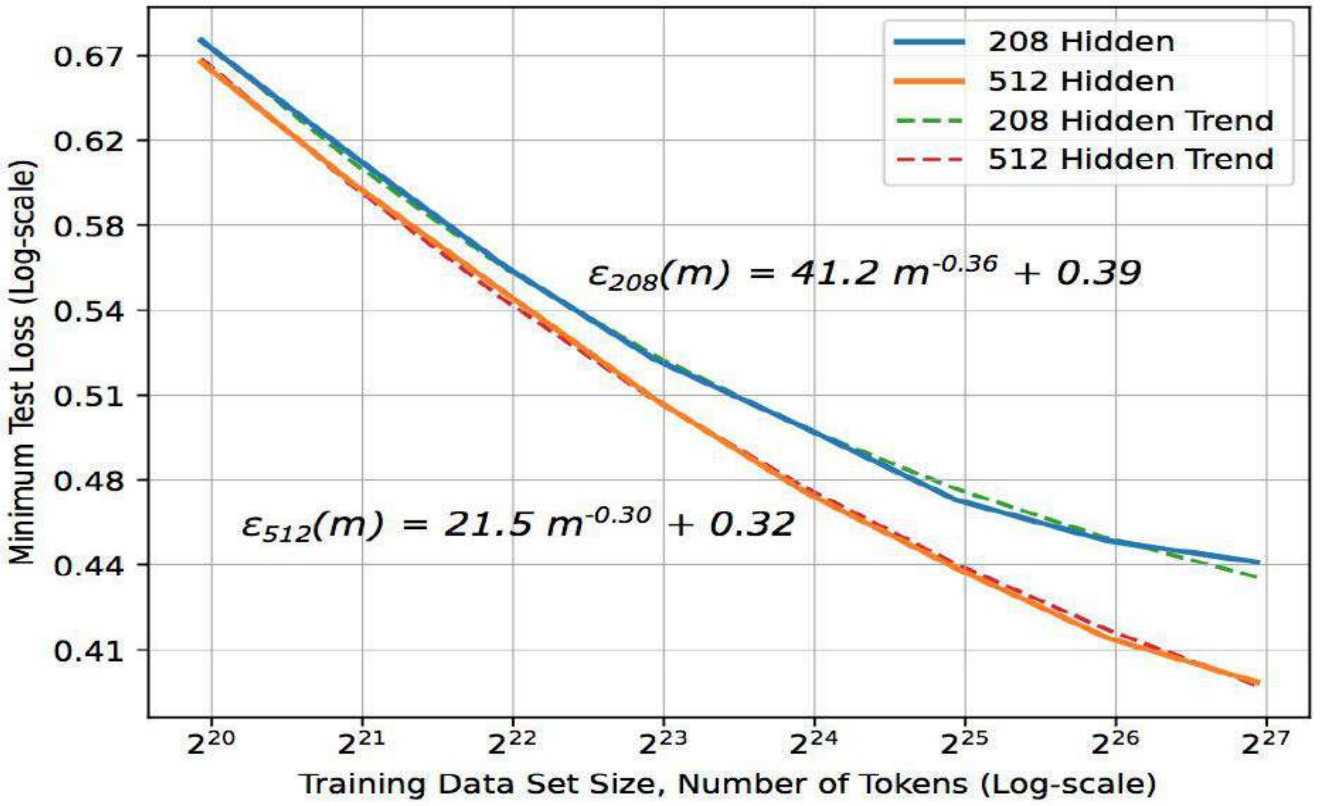

Edge
TPU



Raspberry Pi

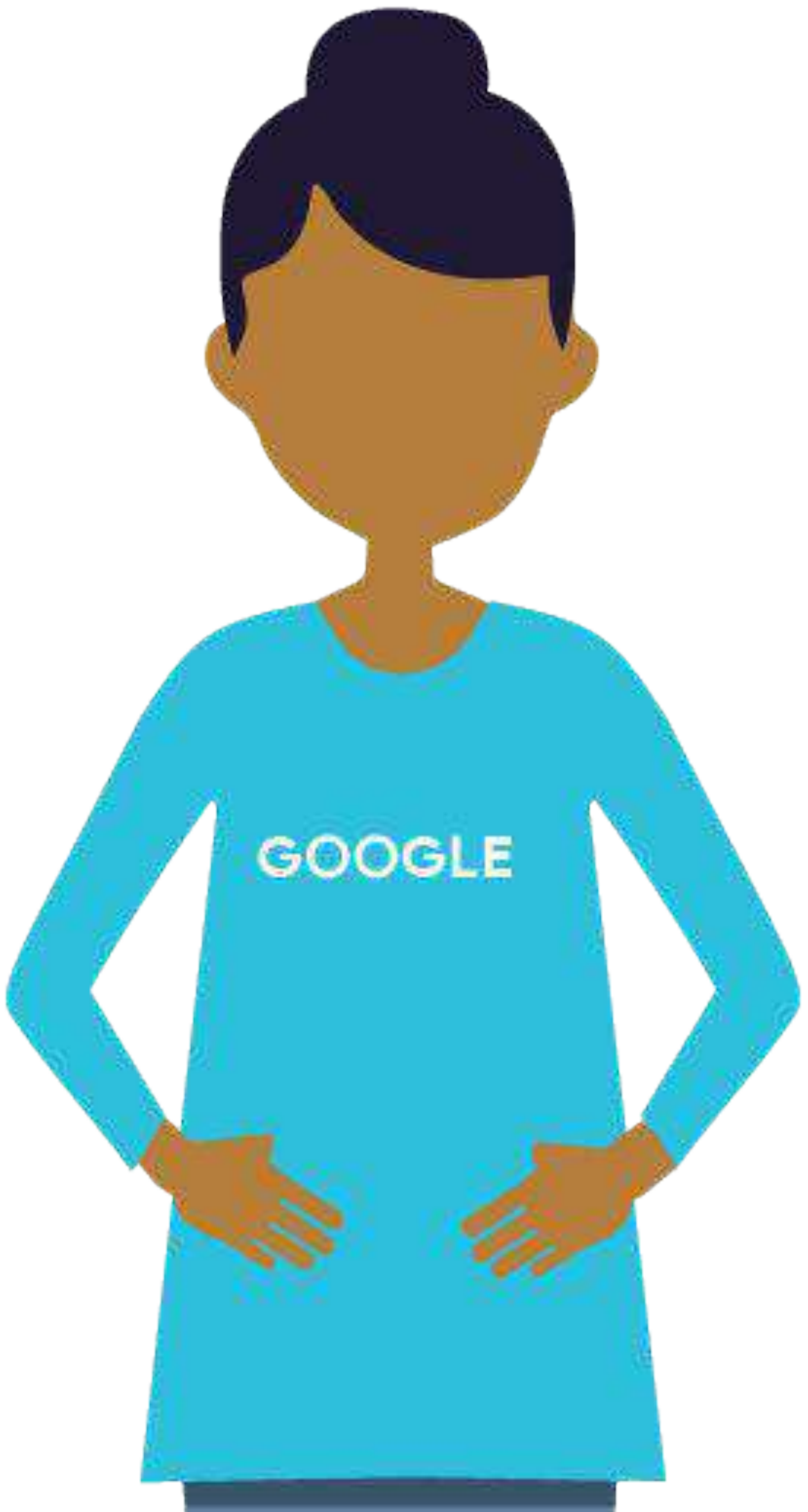



## Deep learning works because datasets are large, but the compute required keeps increasing

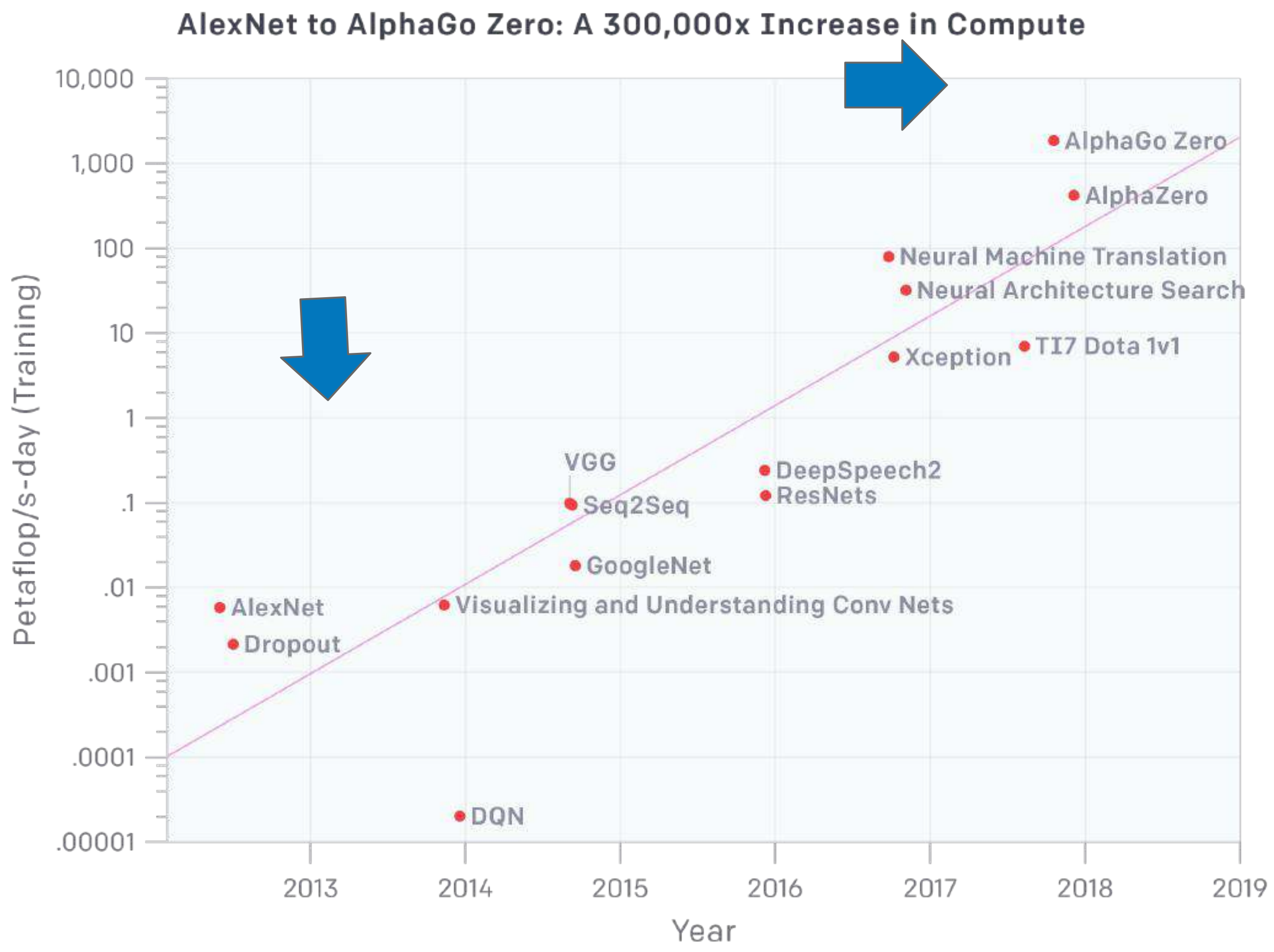



The unreasonable effectiveness of data
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35179.pdf

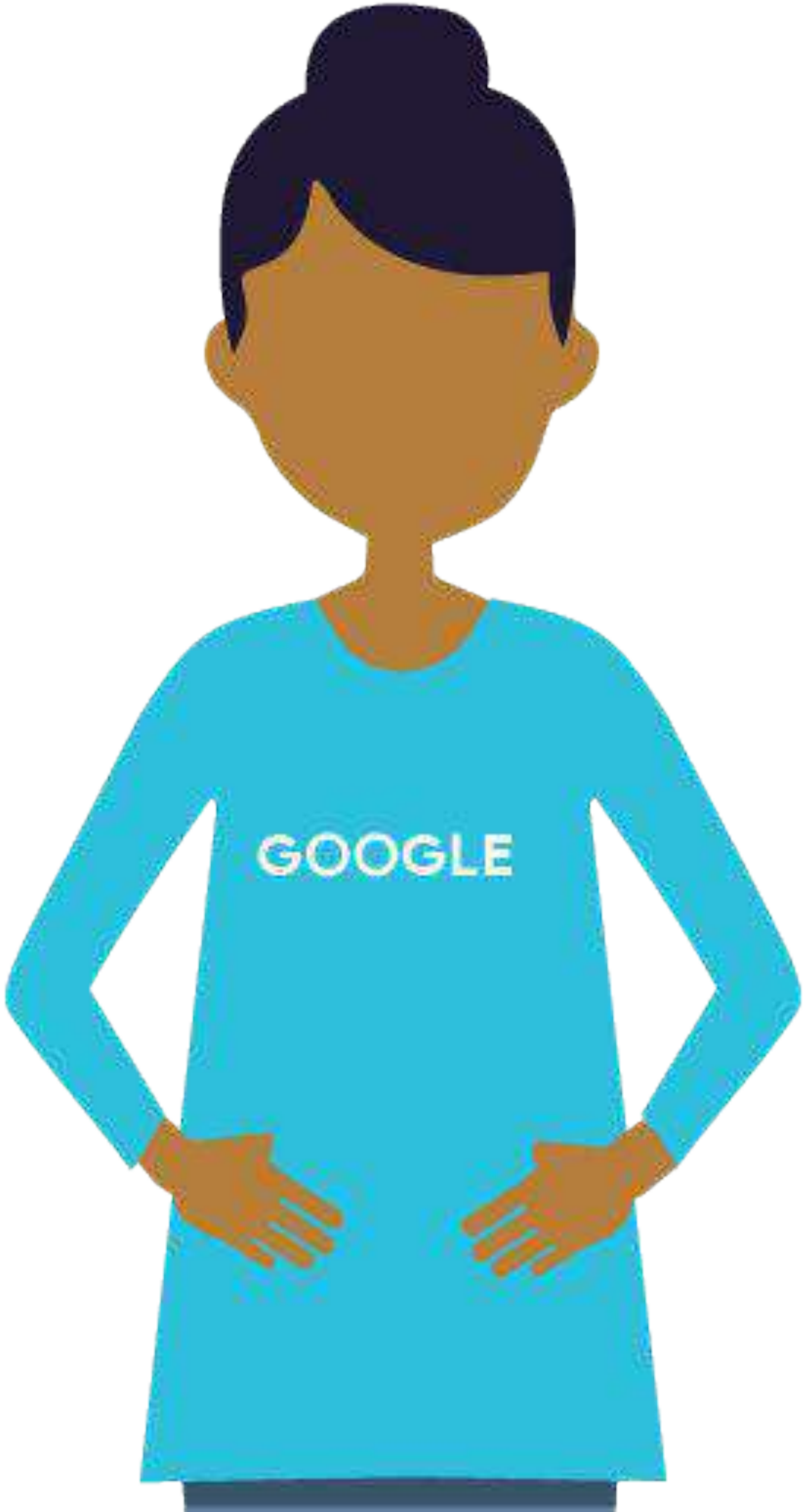
Deep Learning scaling is predictable, empirically
https://arxiv.org/abs/1712.00409



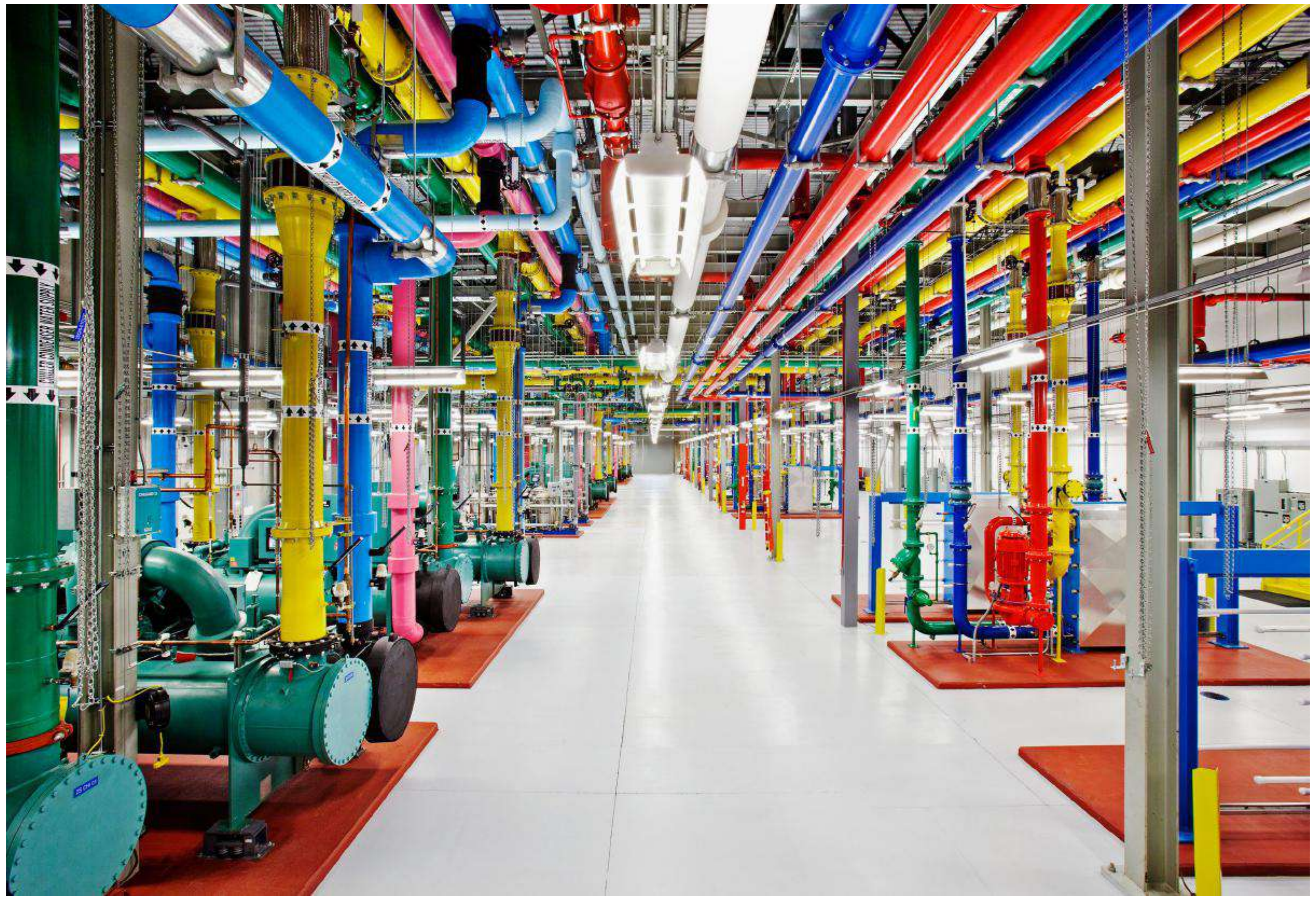

Deep learning works because datasets are large, but the compute required keeps increasing



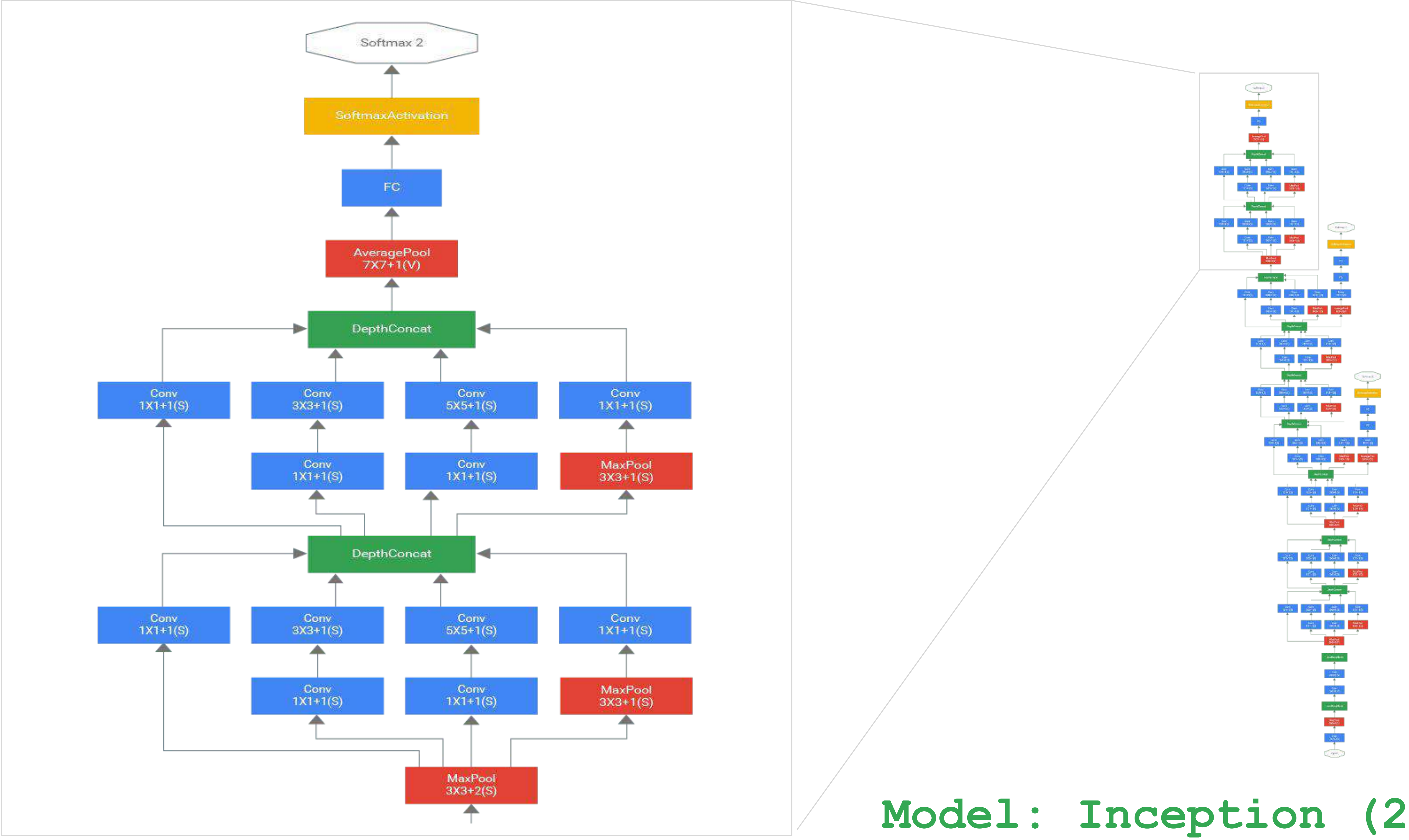

https://blog.openai.com/ai-and-compute/




Distributed systems are a necessity for managing complex models with large data volumes

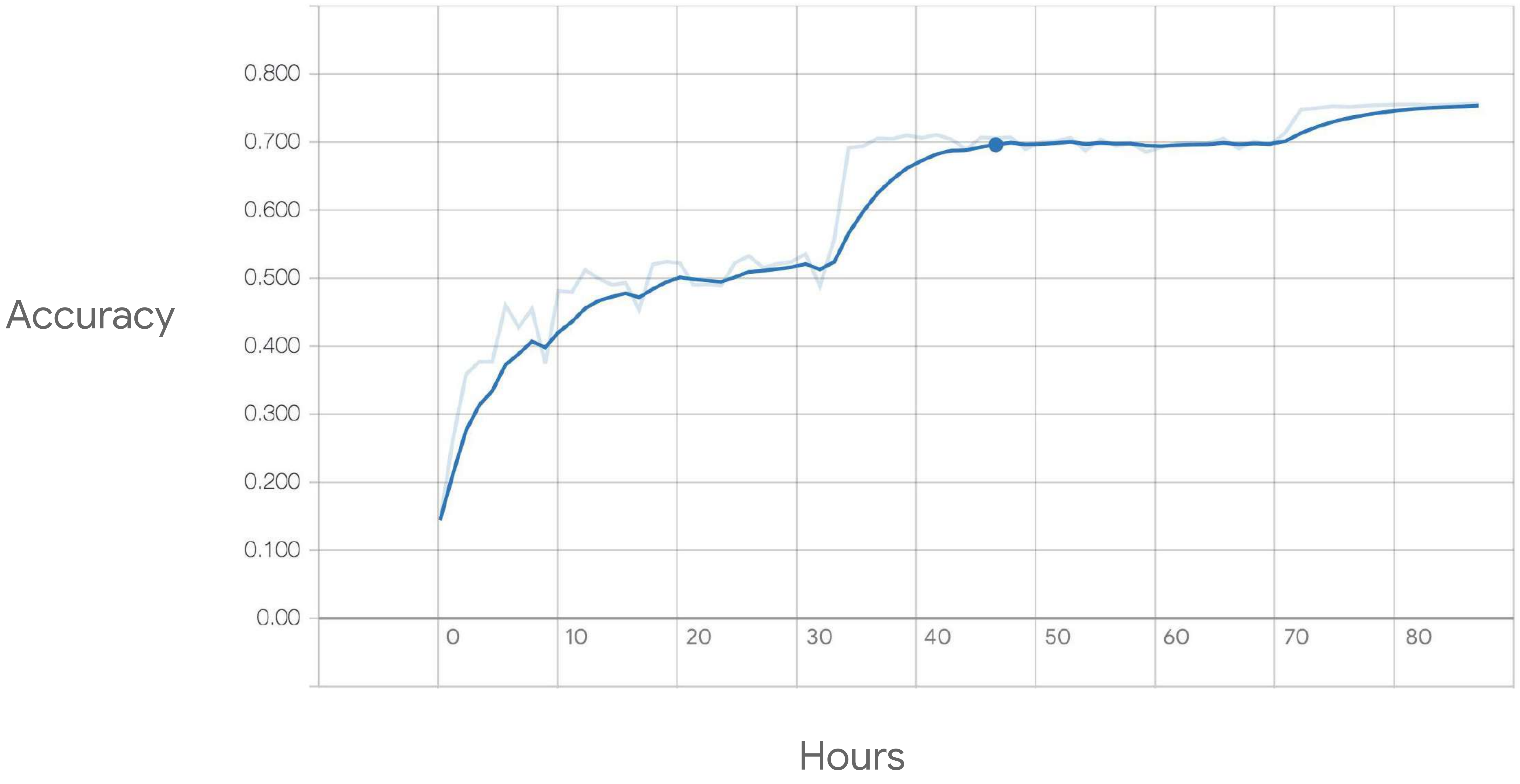


# Large models could have millions of weights

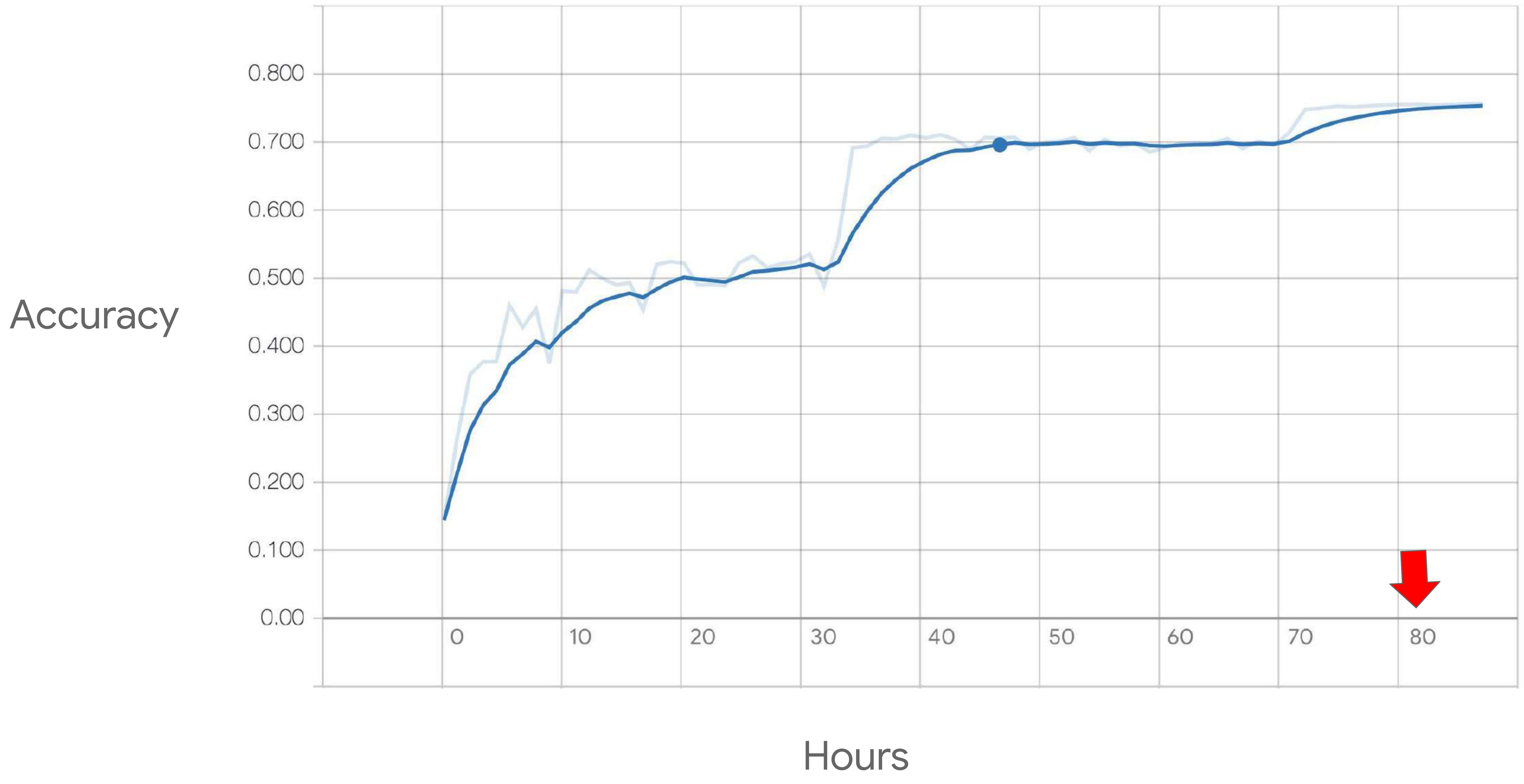


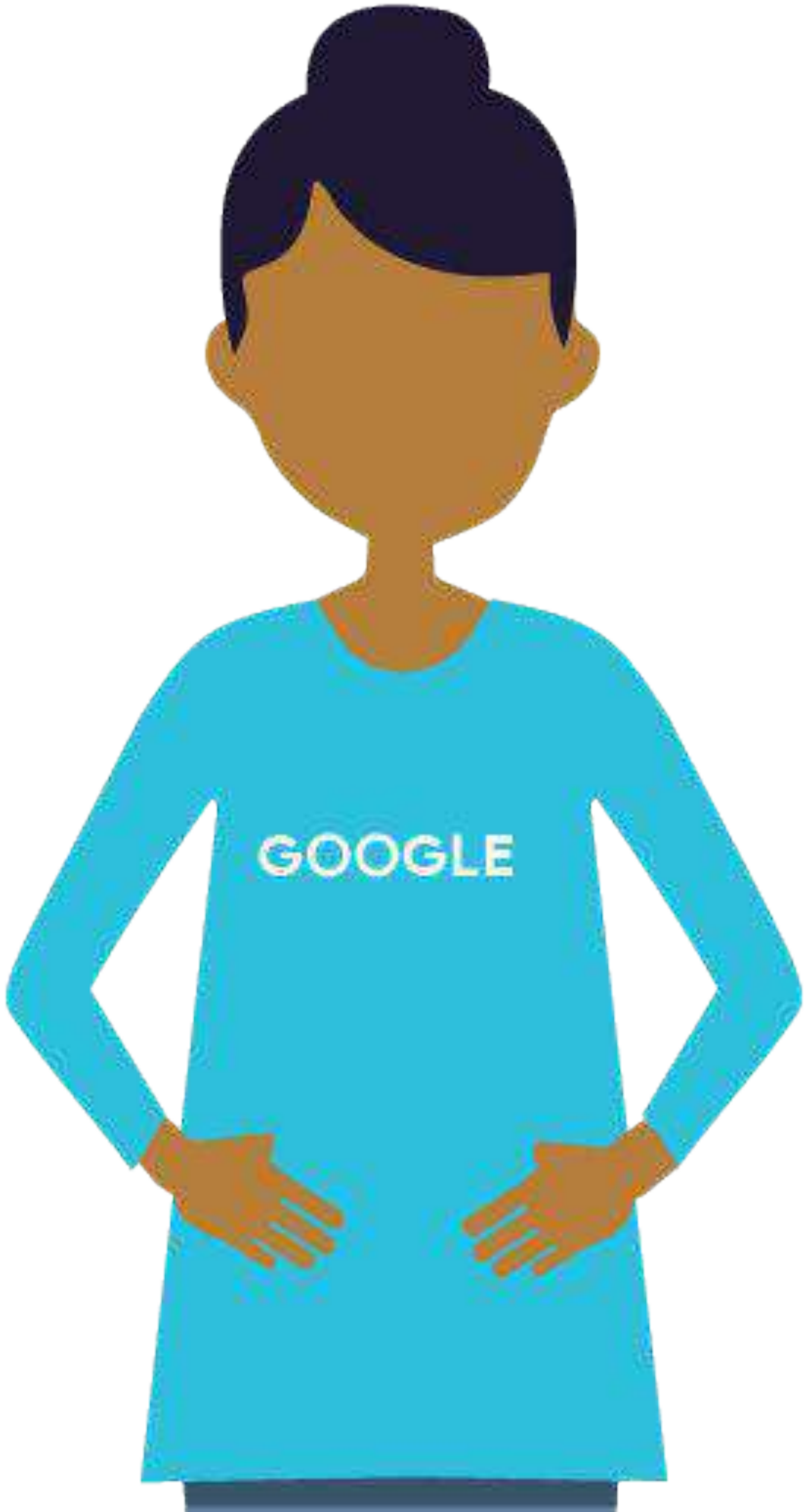

Model: Inception (2015)

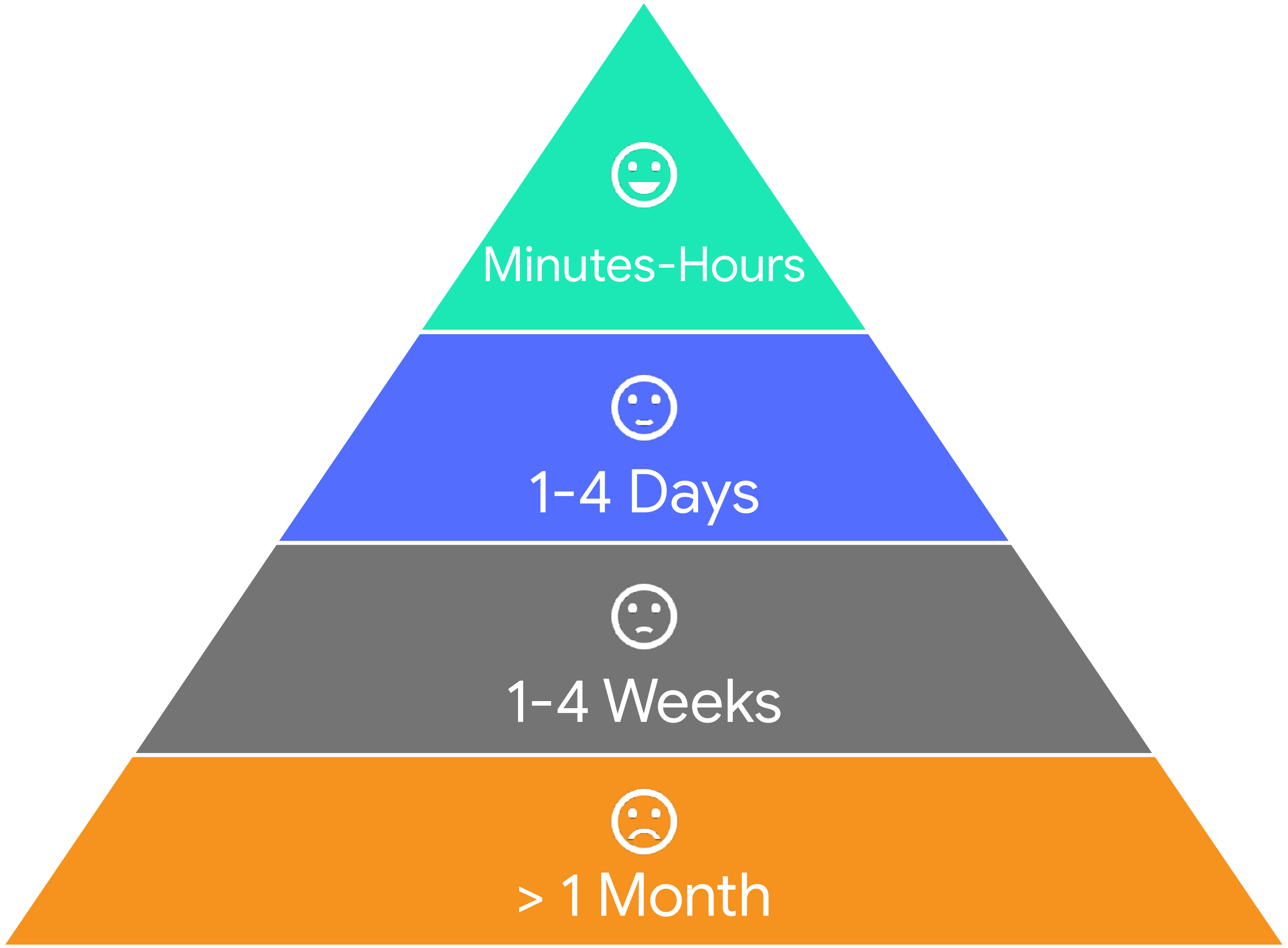
# Training can take a long time

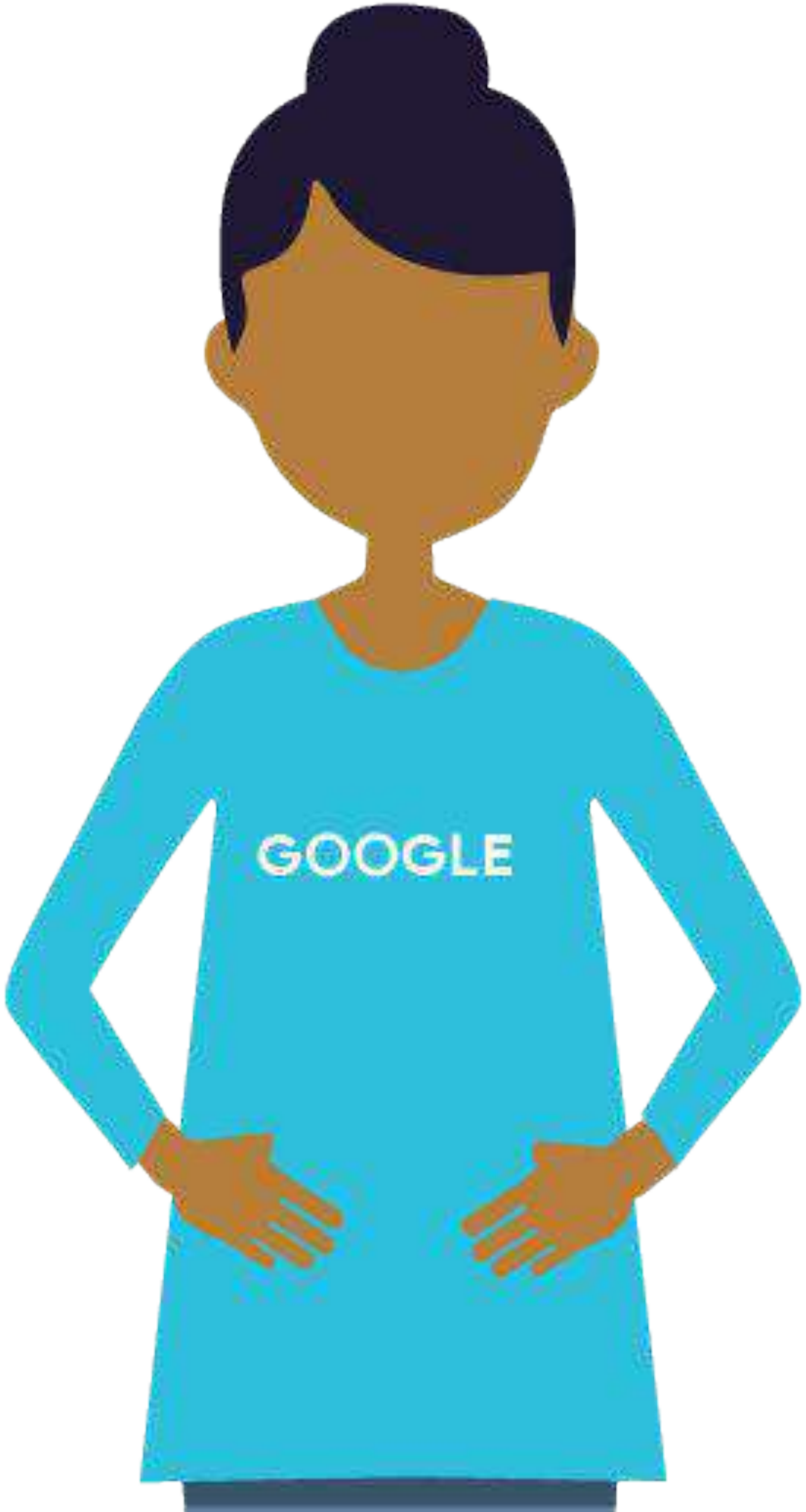


# Training can take a long time

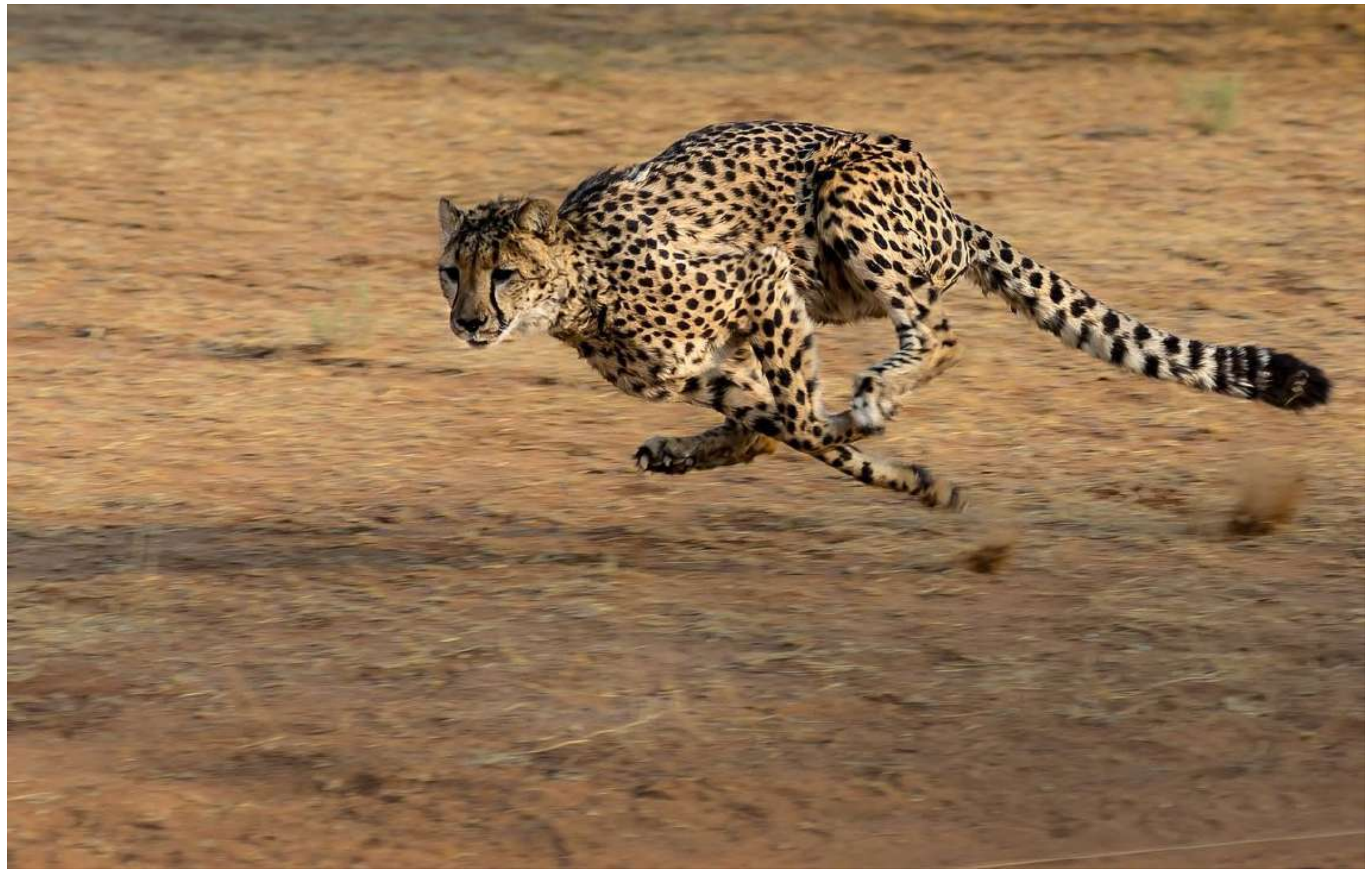


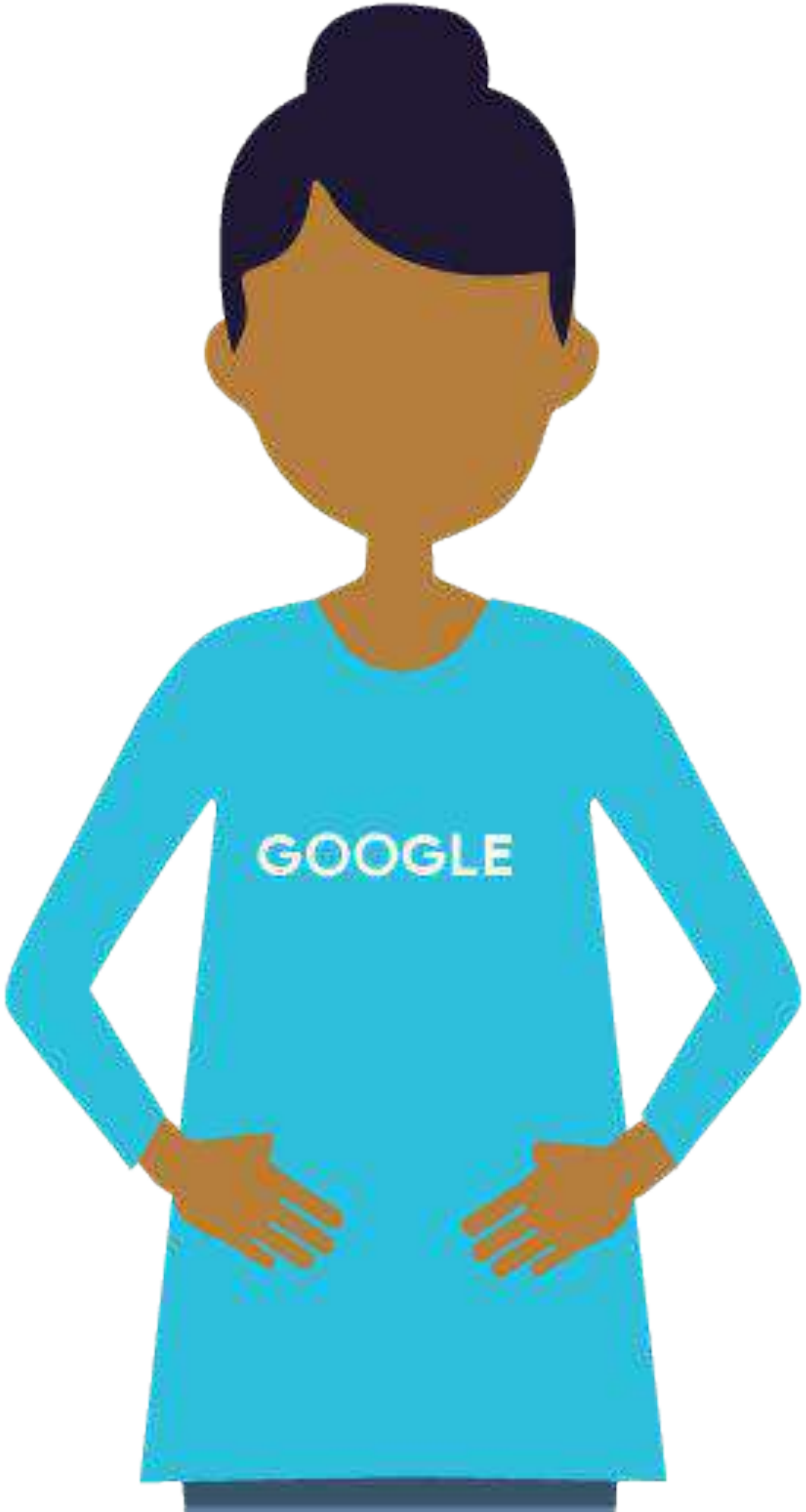

GOOGLE


Minutes-Hours
1-4 Days
1-4 Weeks
> 1 Month

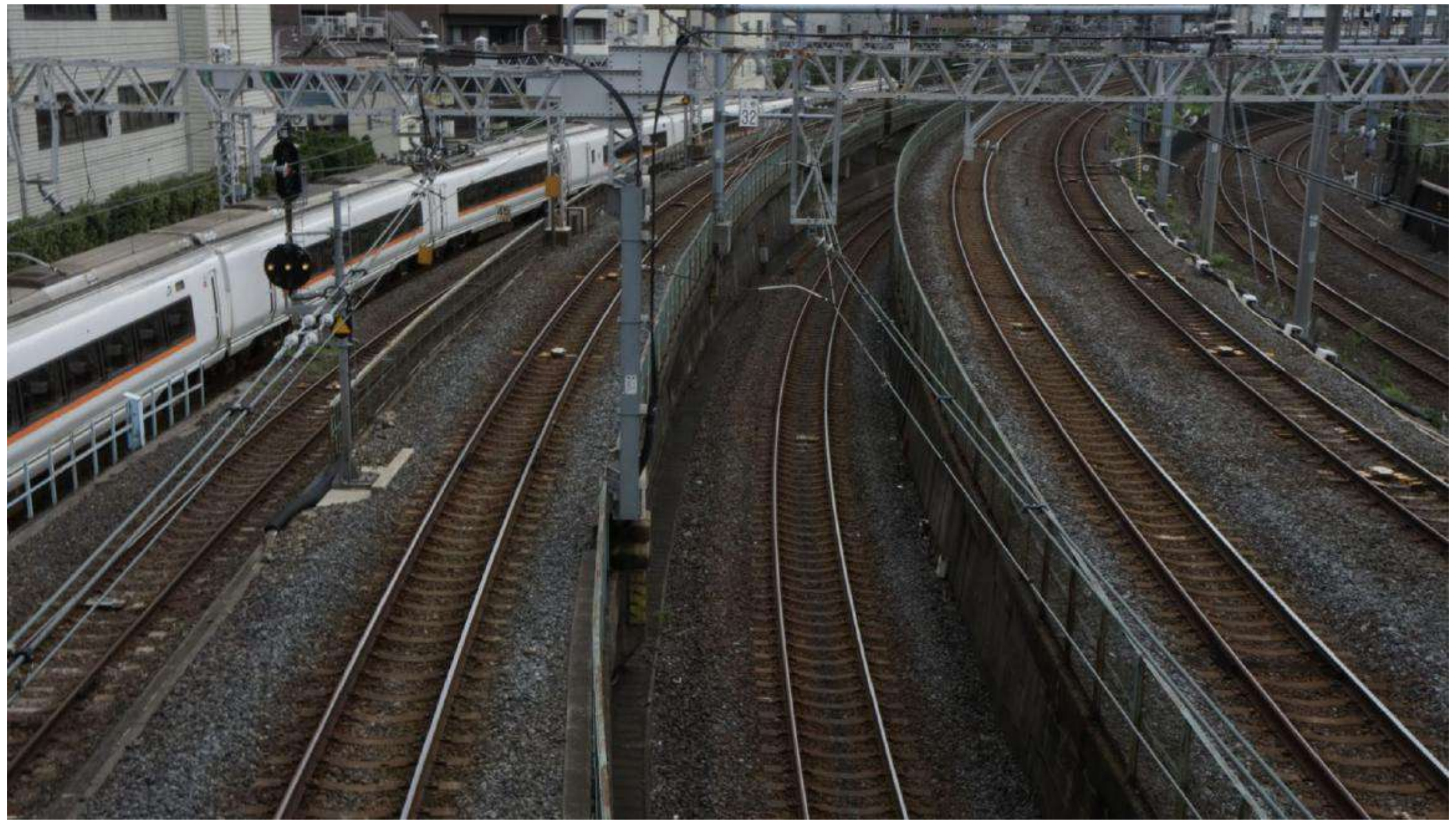


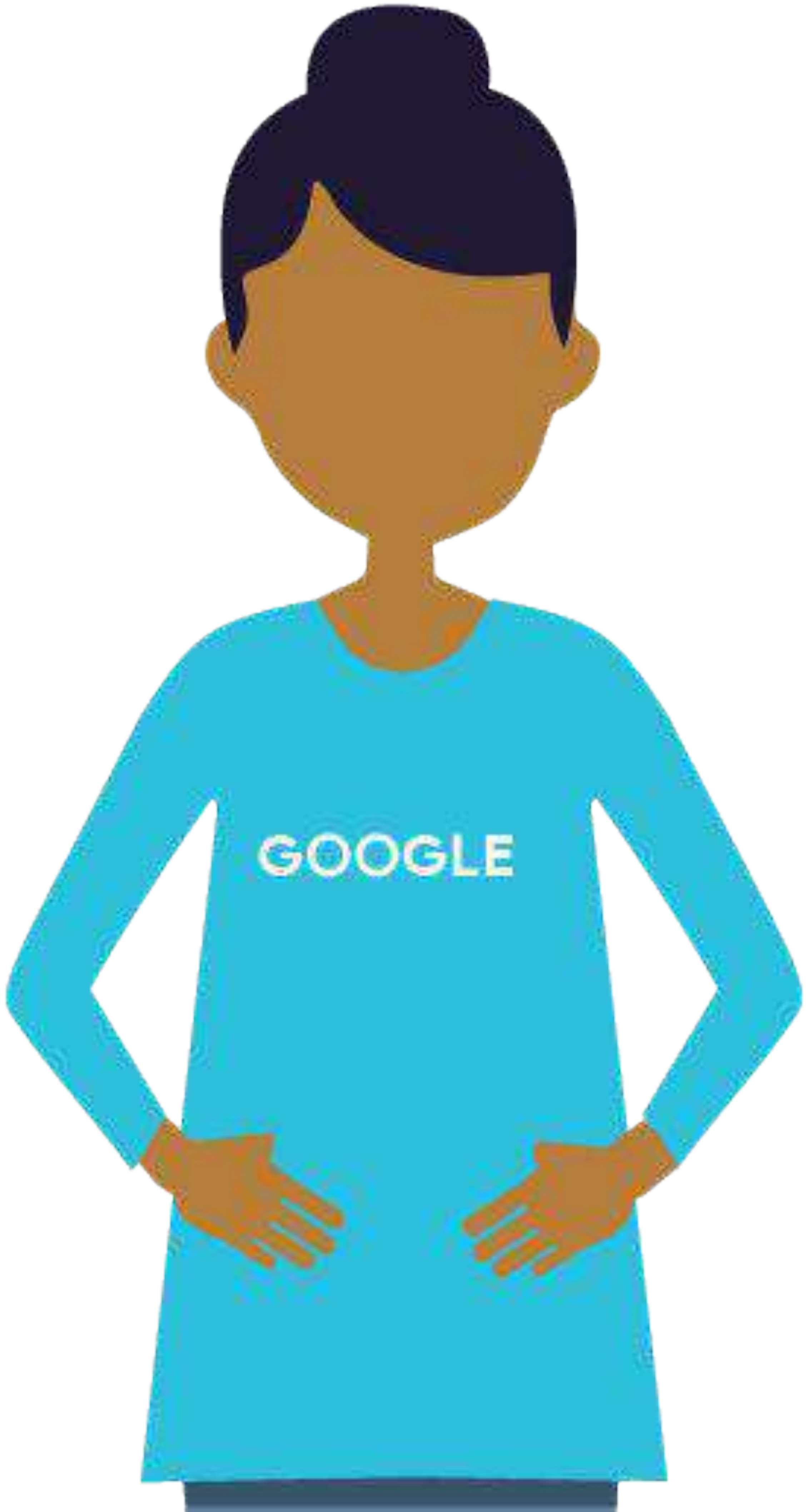

How can you make model training faster?



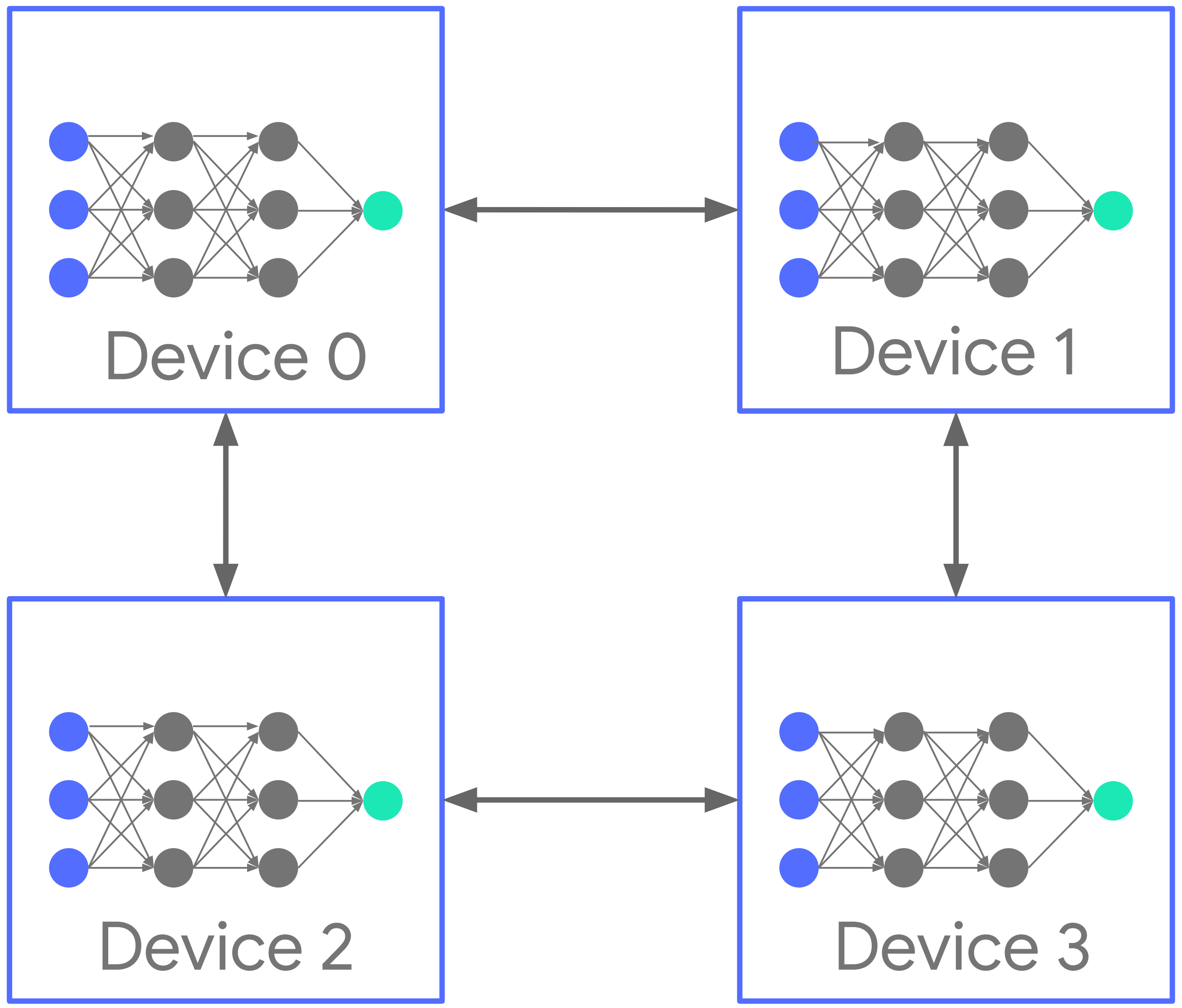


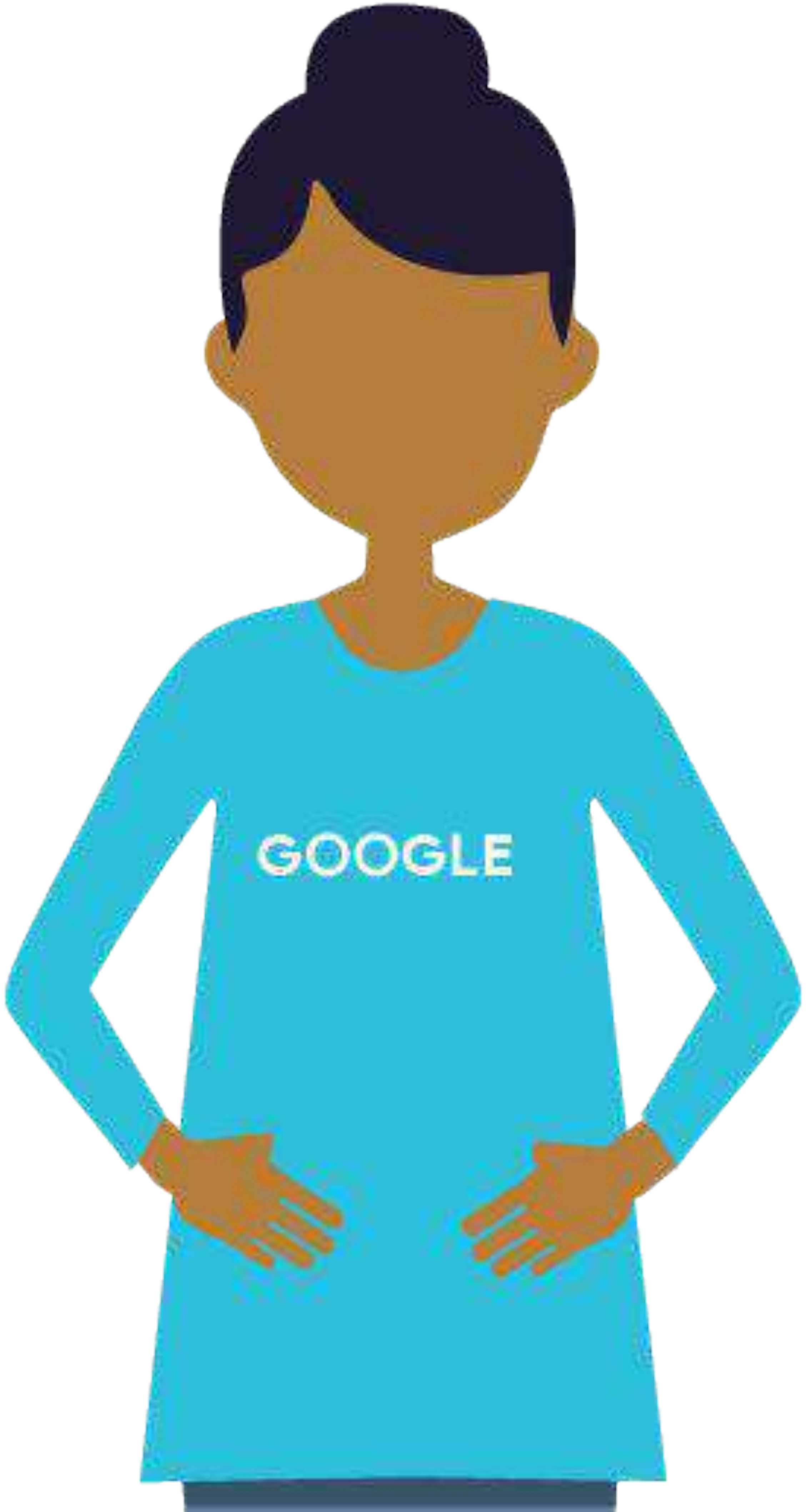

Gaining speed through parallel training



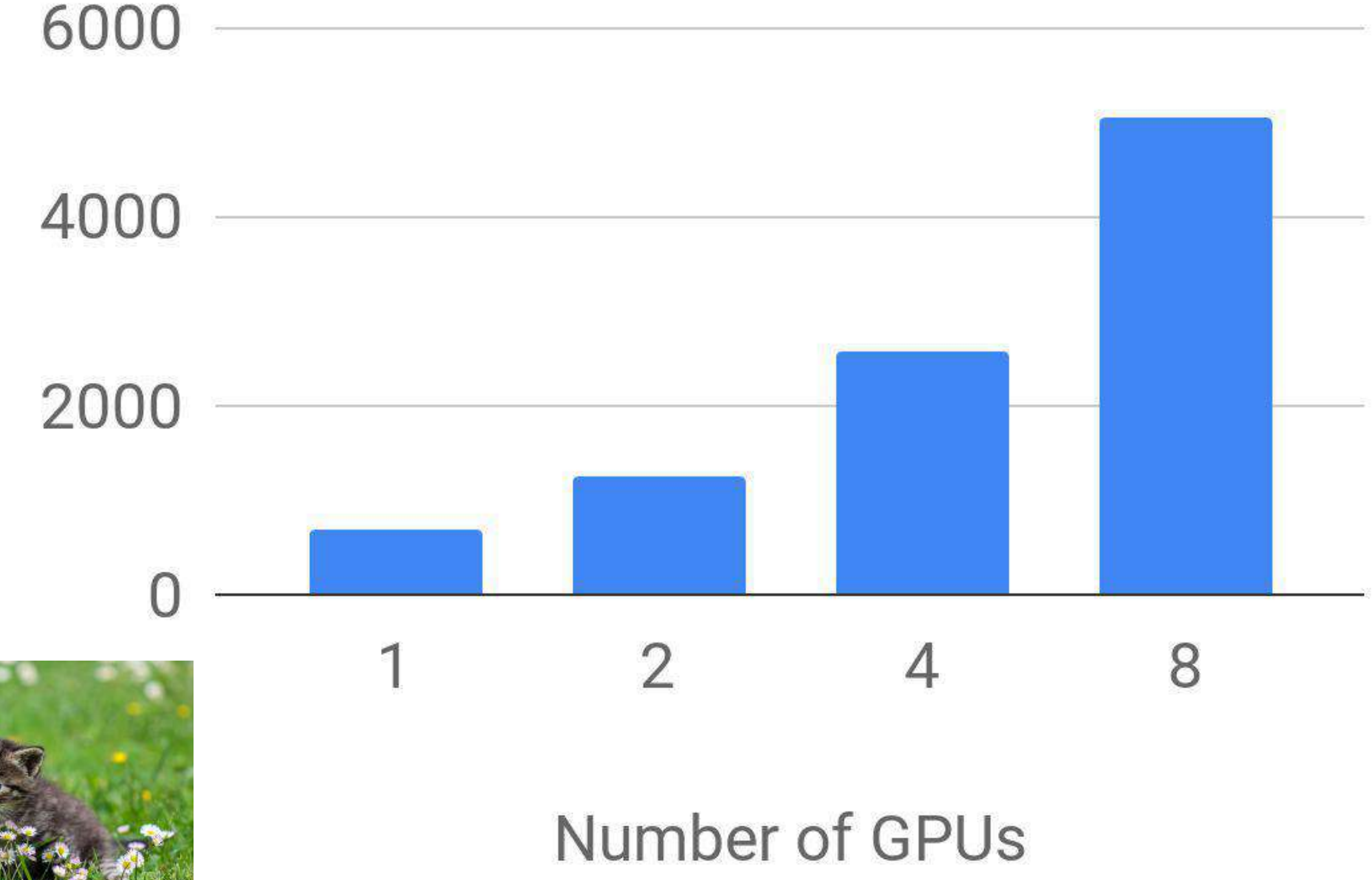



# Gaining speed through parallel training






# Scaling with Distributed Training




Courses 7 - Production ML Systems

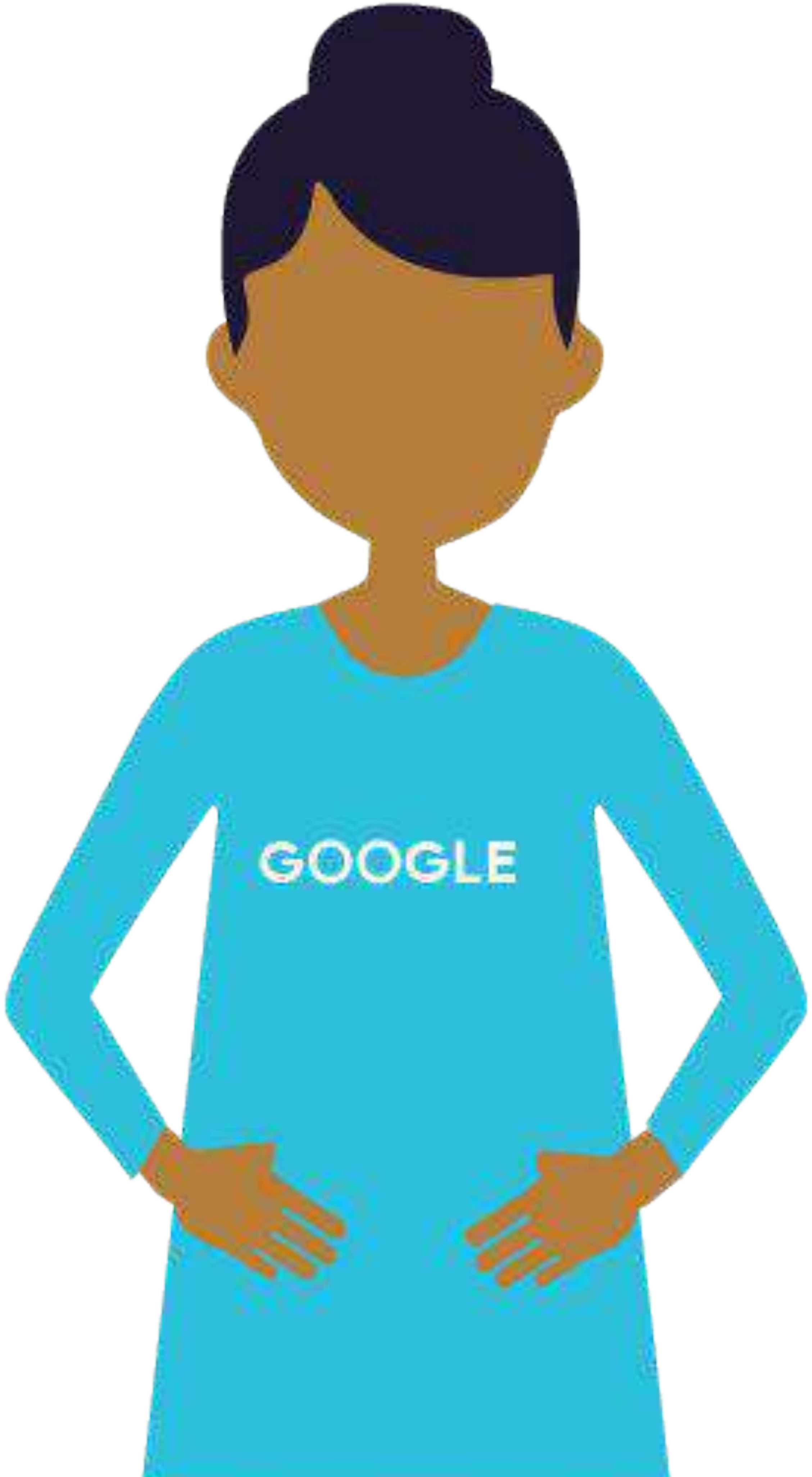
Module 4: Designing High-Performance ML Systems

Lesson Title: **Distributed training architectures**

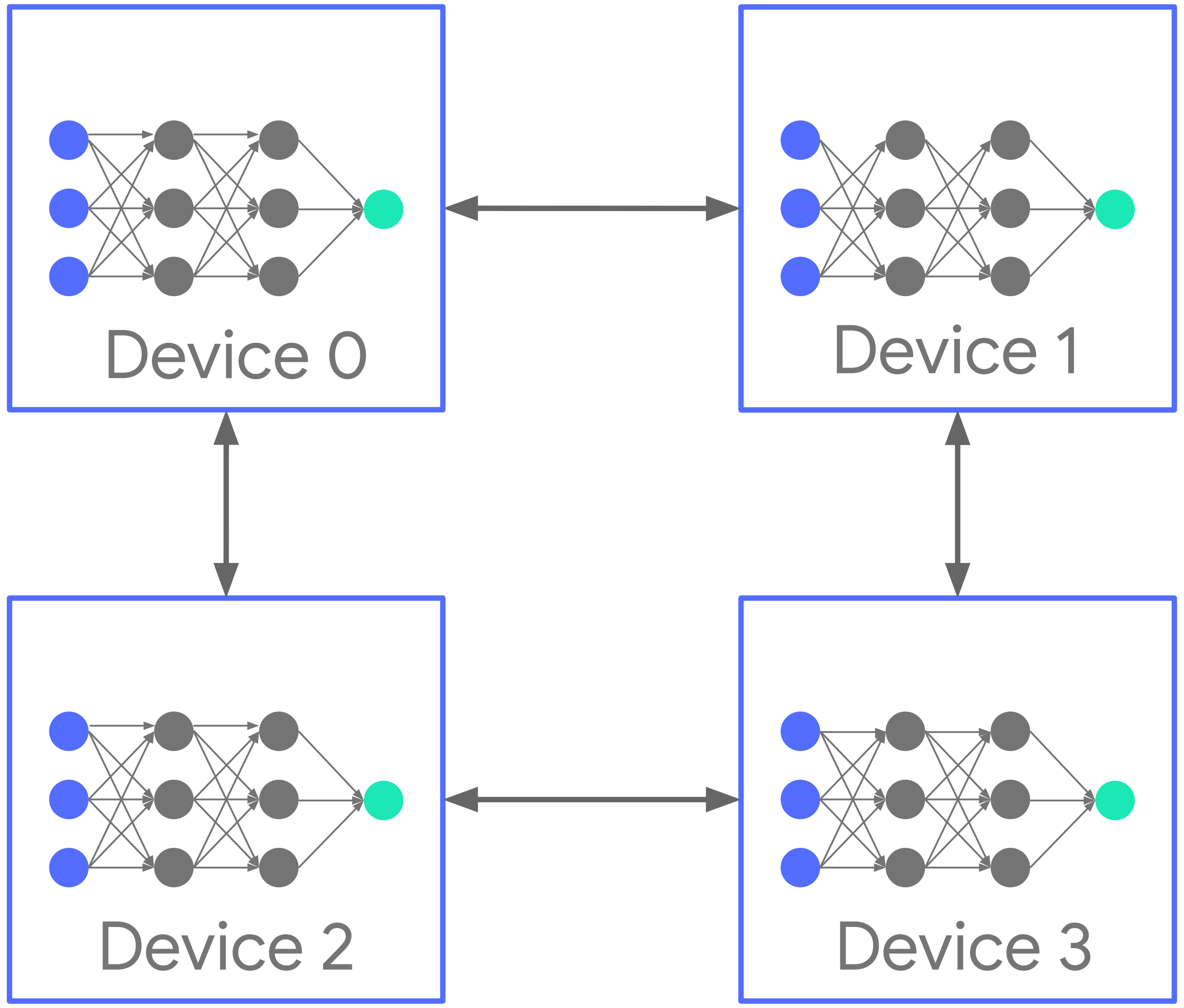
Format: Presenter

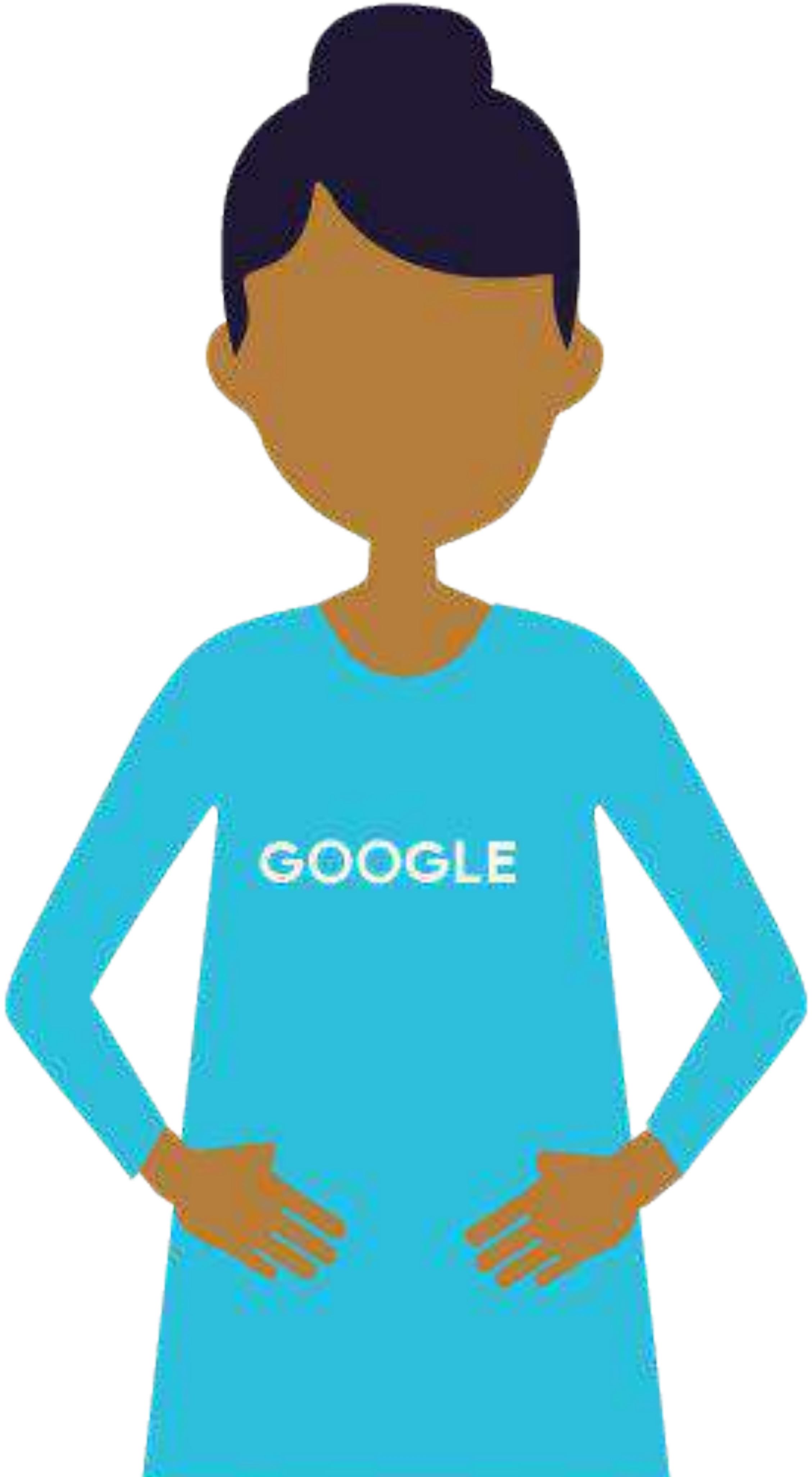
Presenter: Laurence Moroney

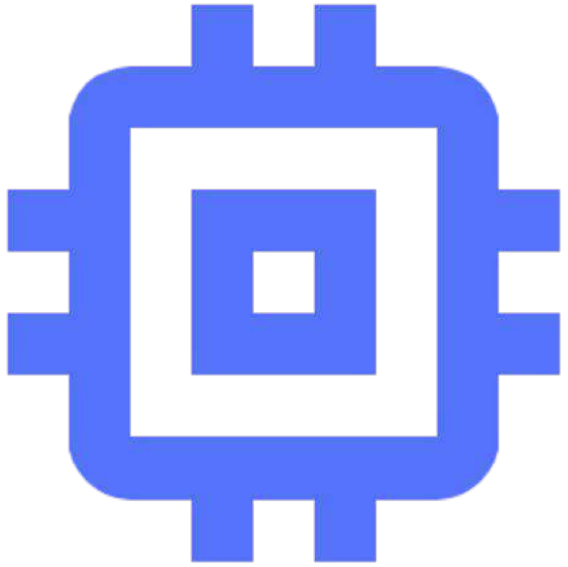
Video Name: T-PSML-O_4_l5_distributed_training_architectures




# Distributed Training Architectures

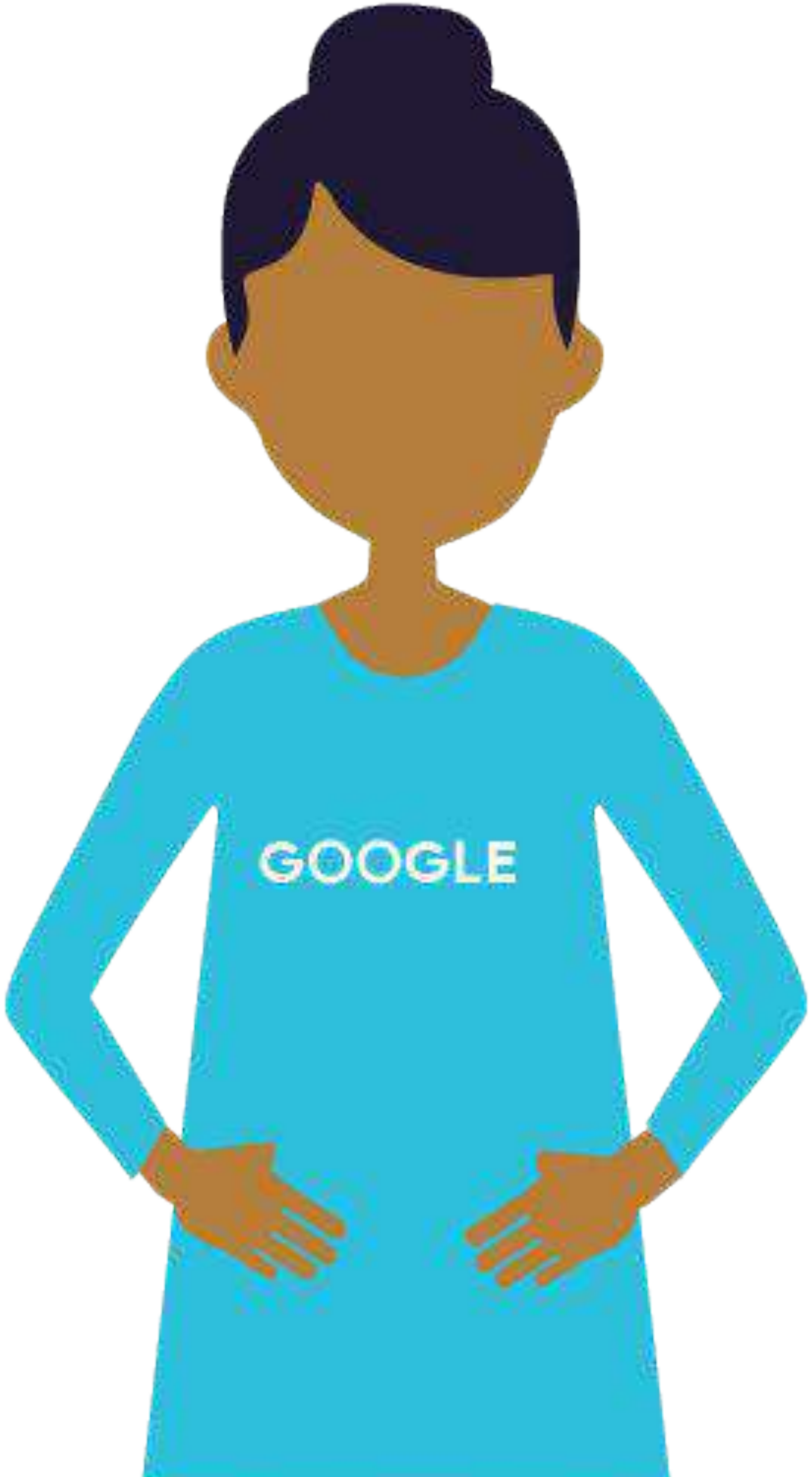


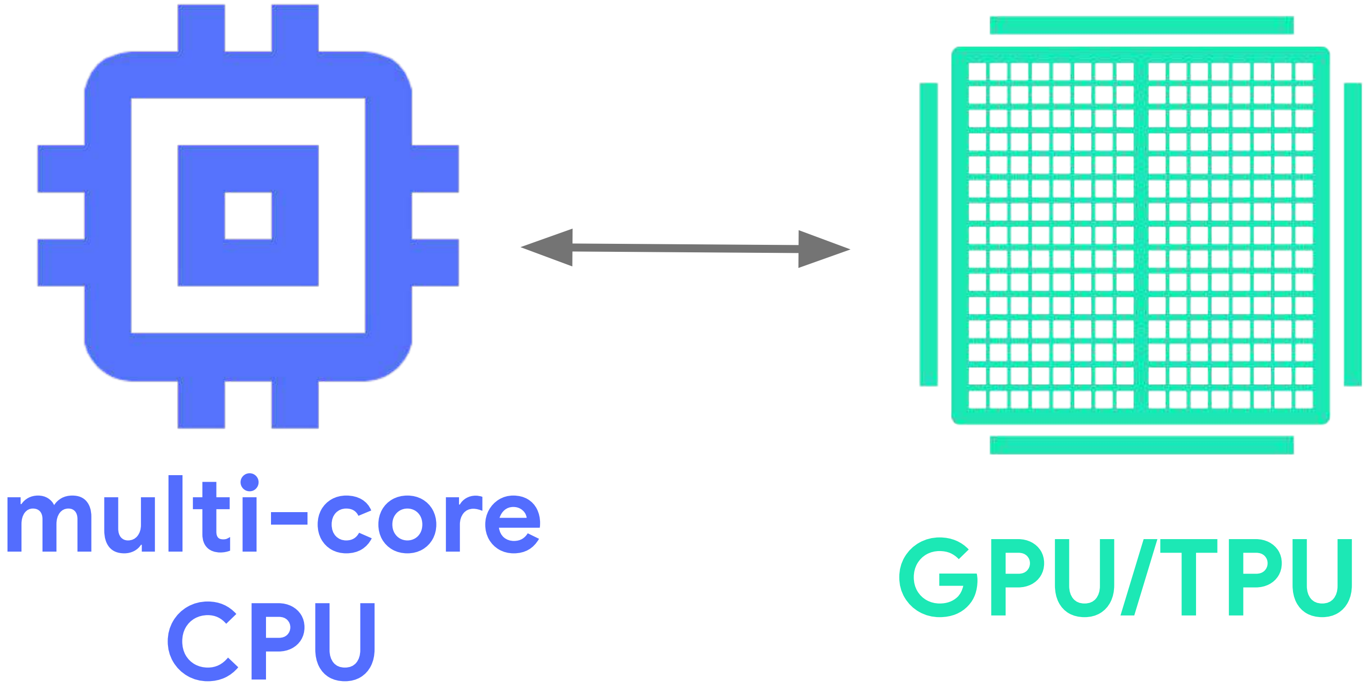


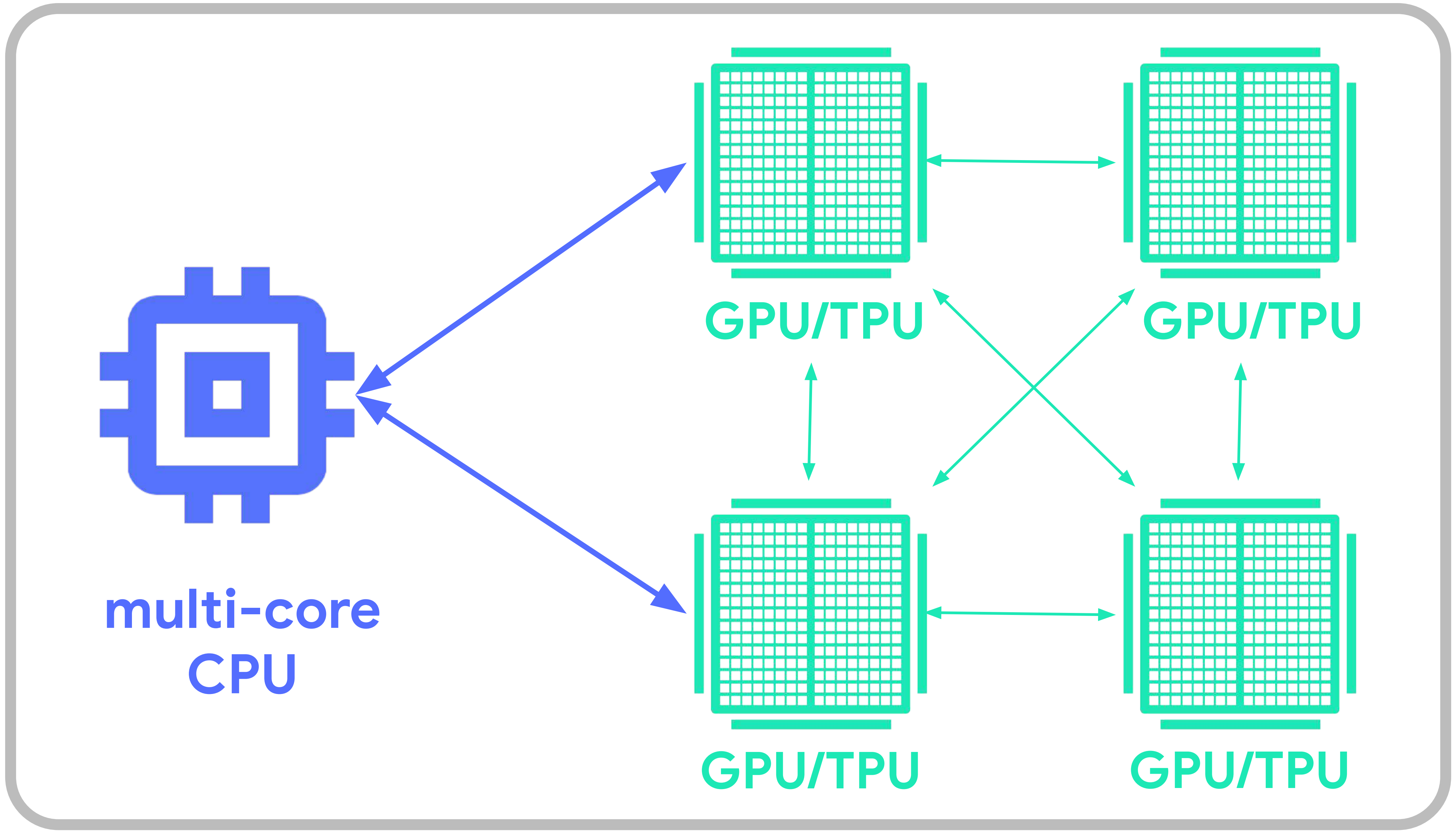
multi-core
CPU

# Adding a single accelerator

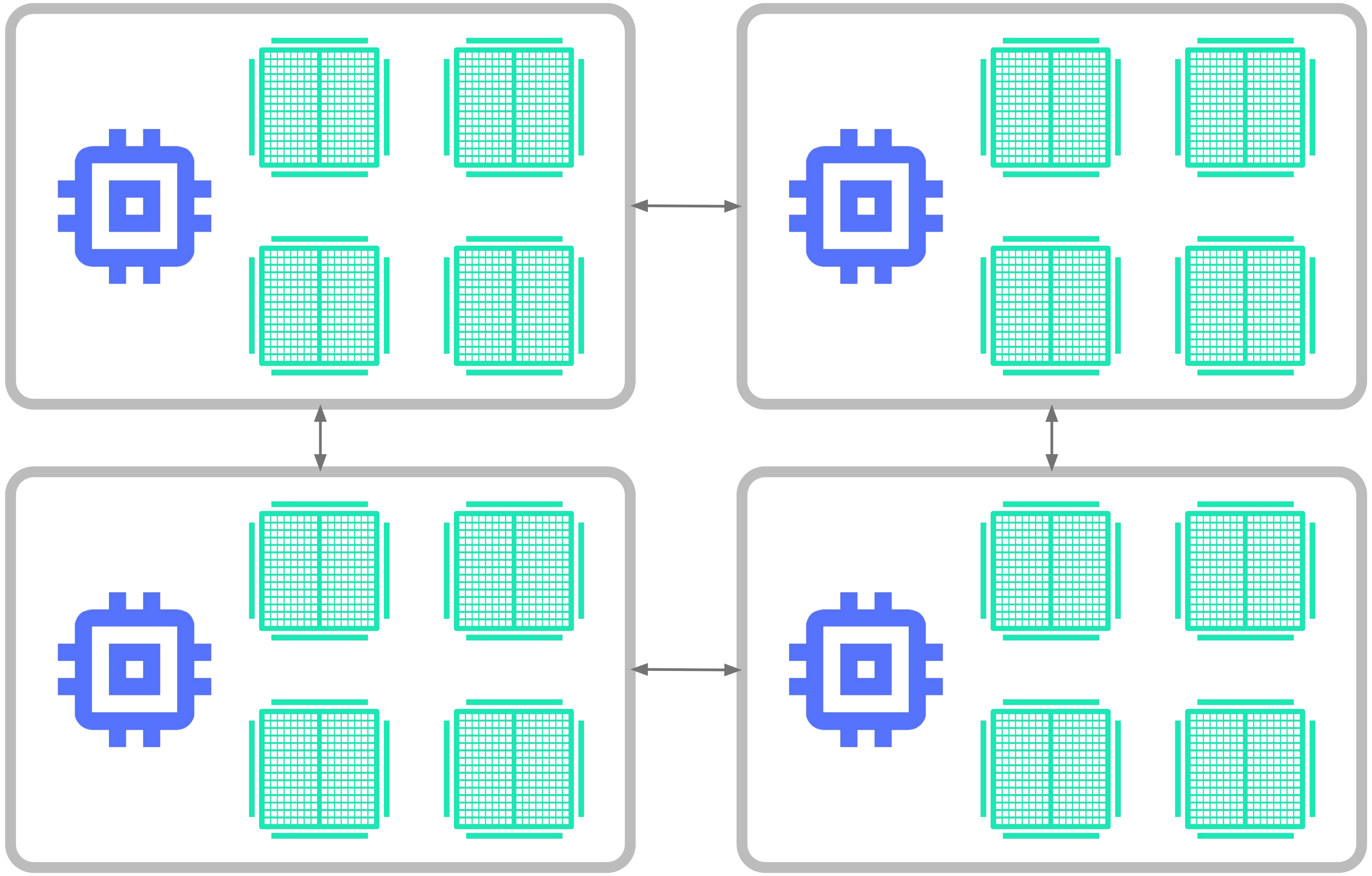


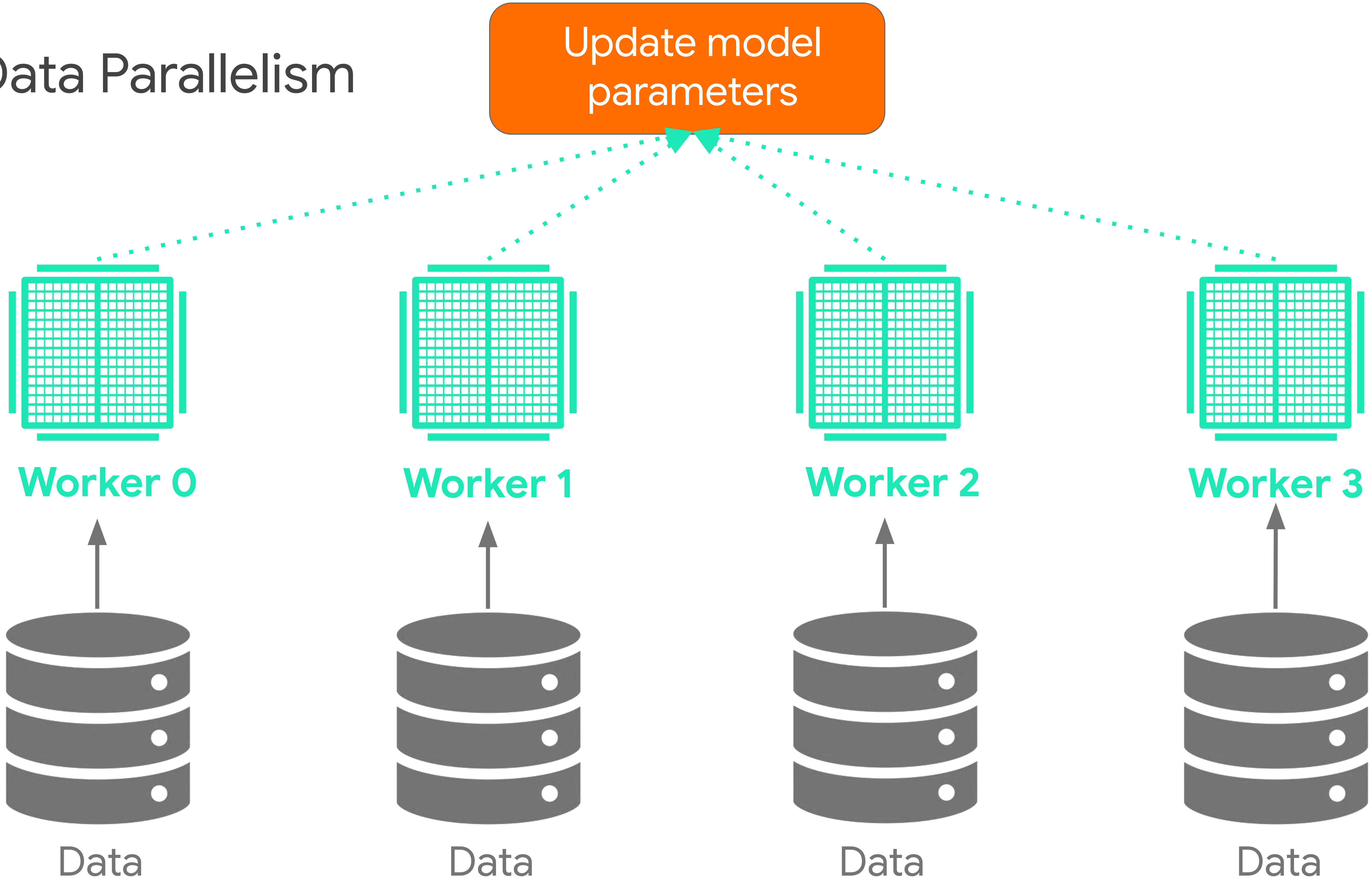
Adding many accelerators to a single device

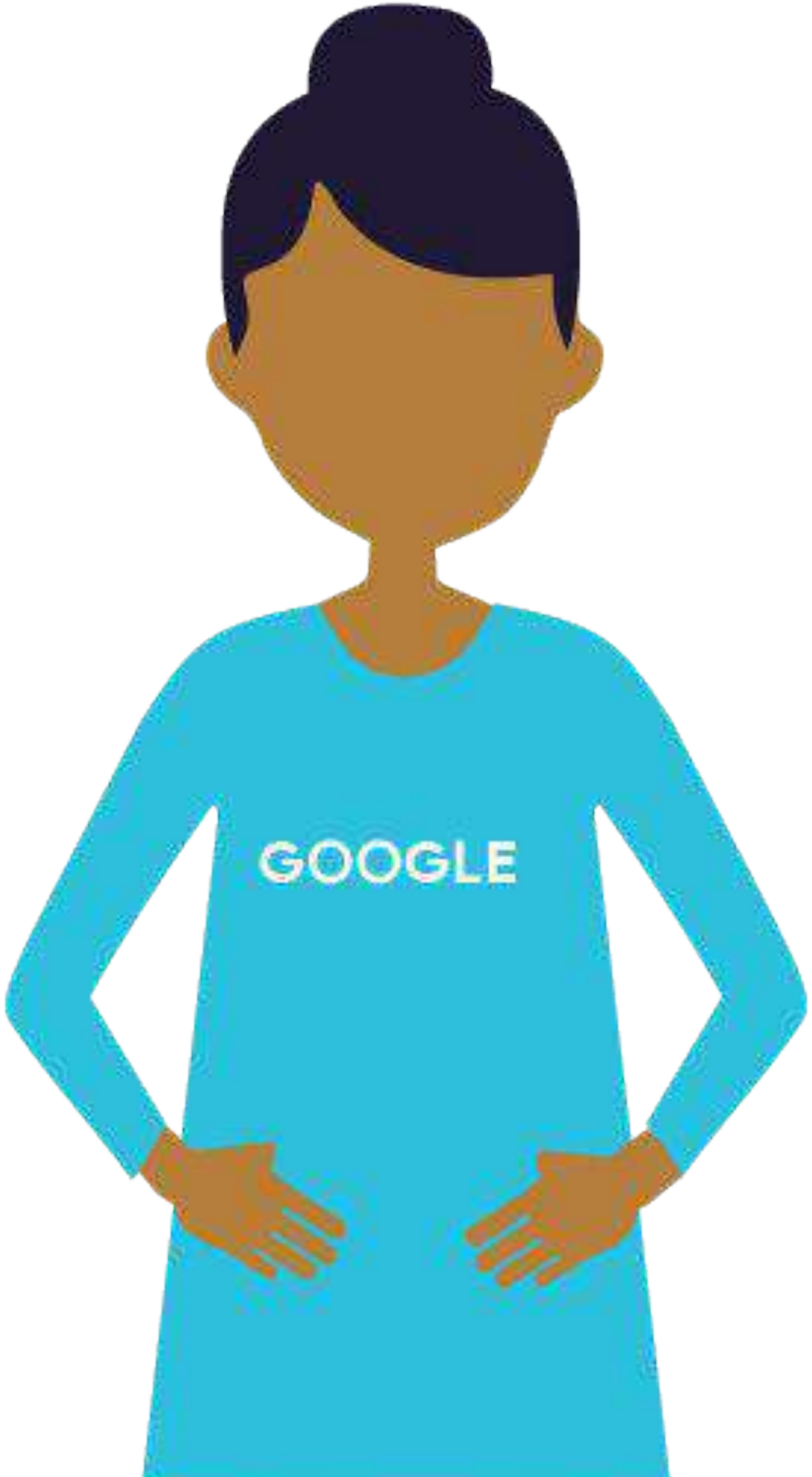


Adding many machines with many possible devices



# Data Parallelism

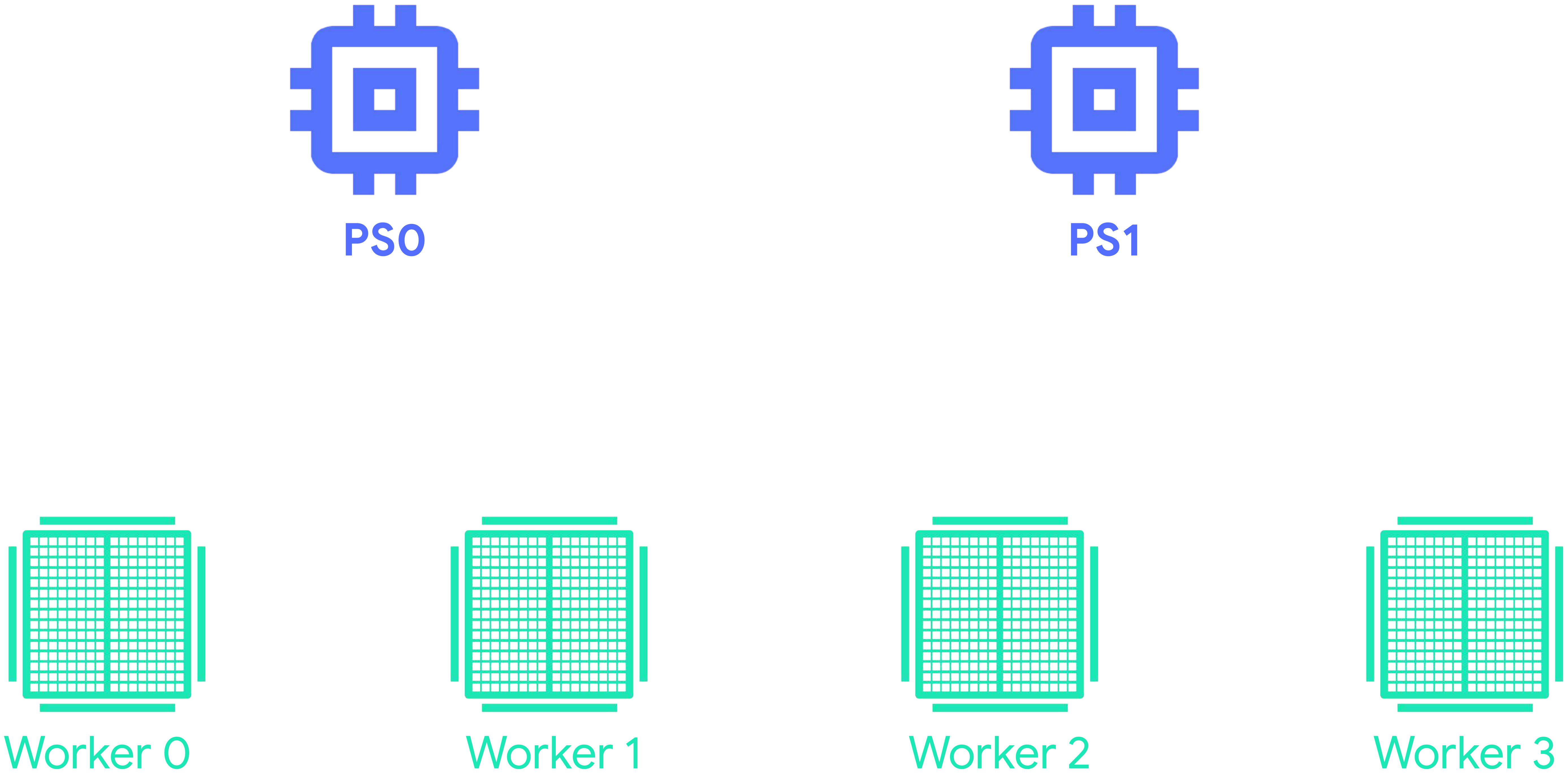




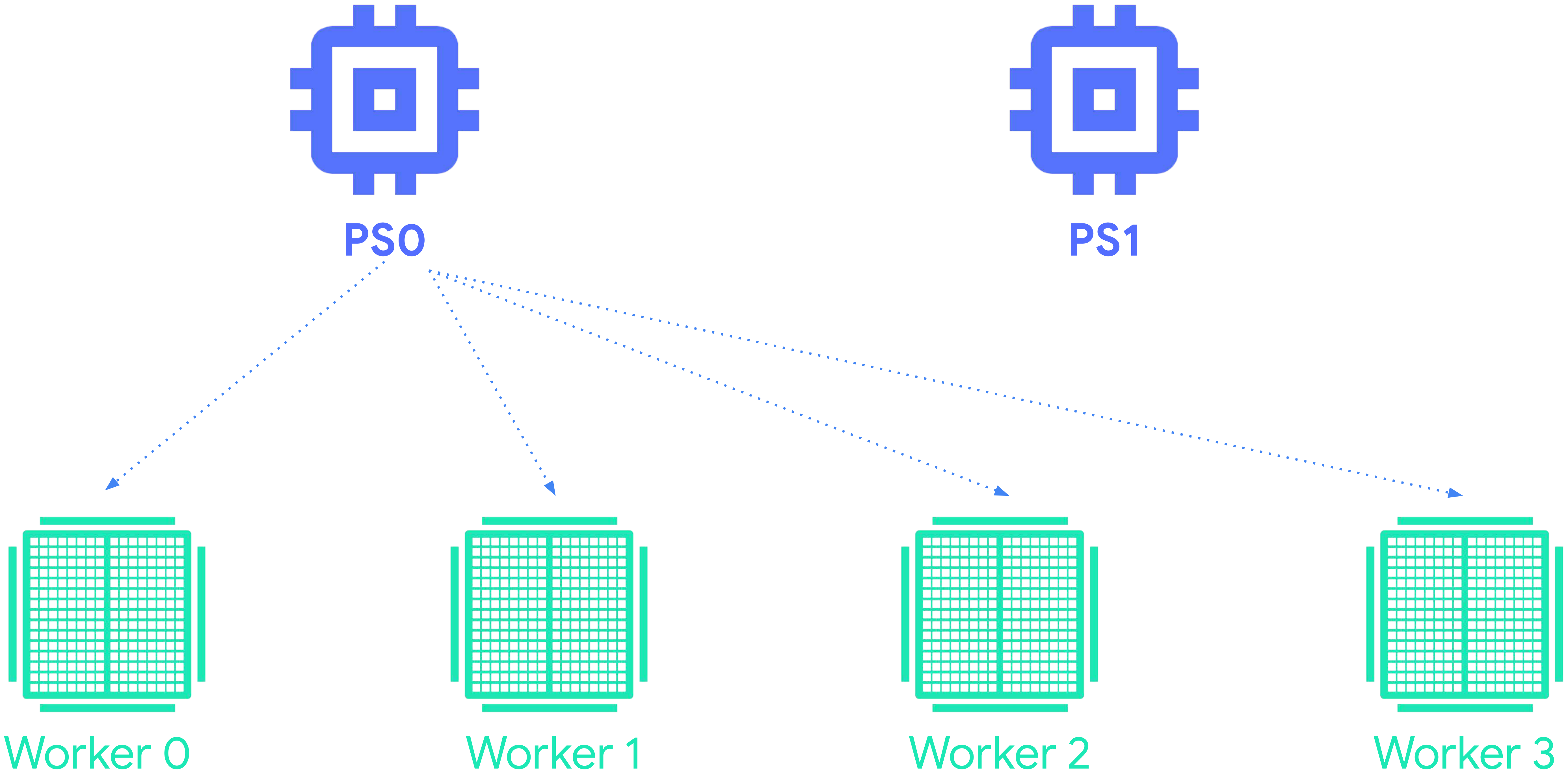

# Two approaches to Data Parallelism

1. Parameter server
2. Sync Allreduce

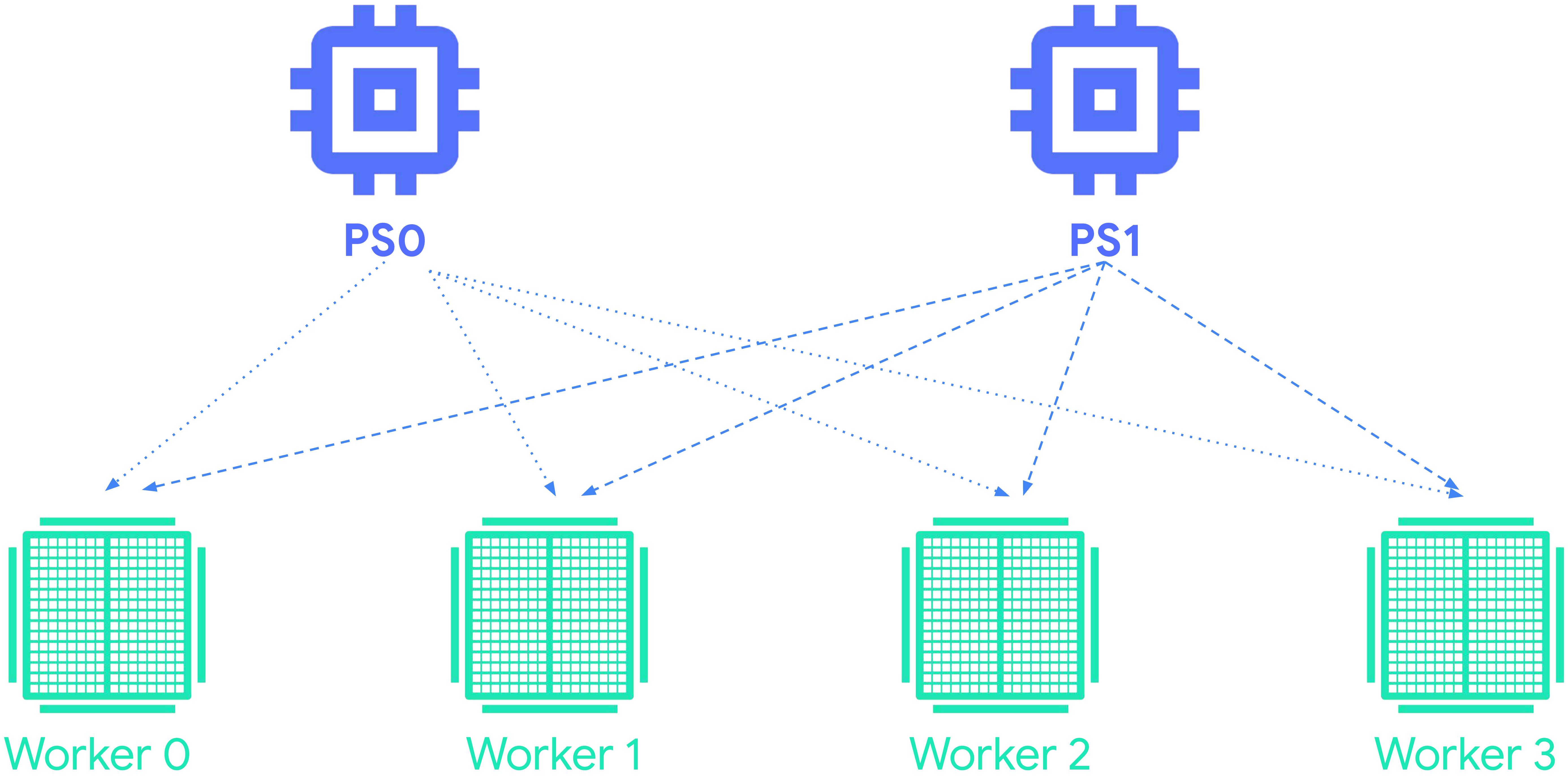
# Async Parameter Server



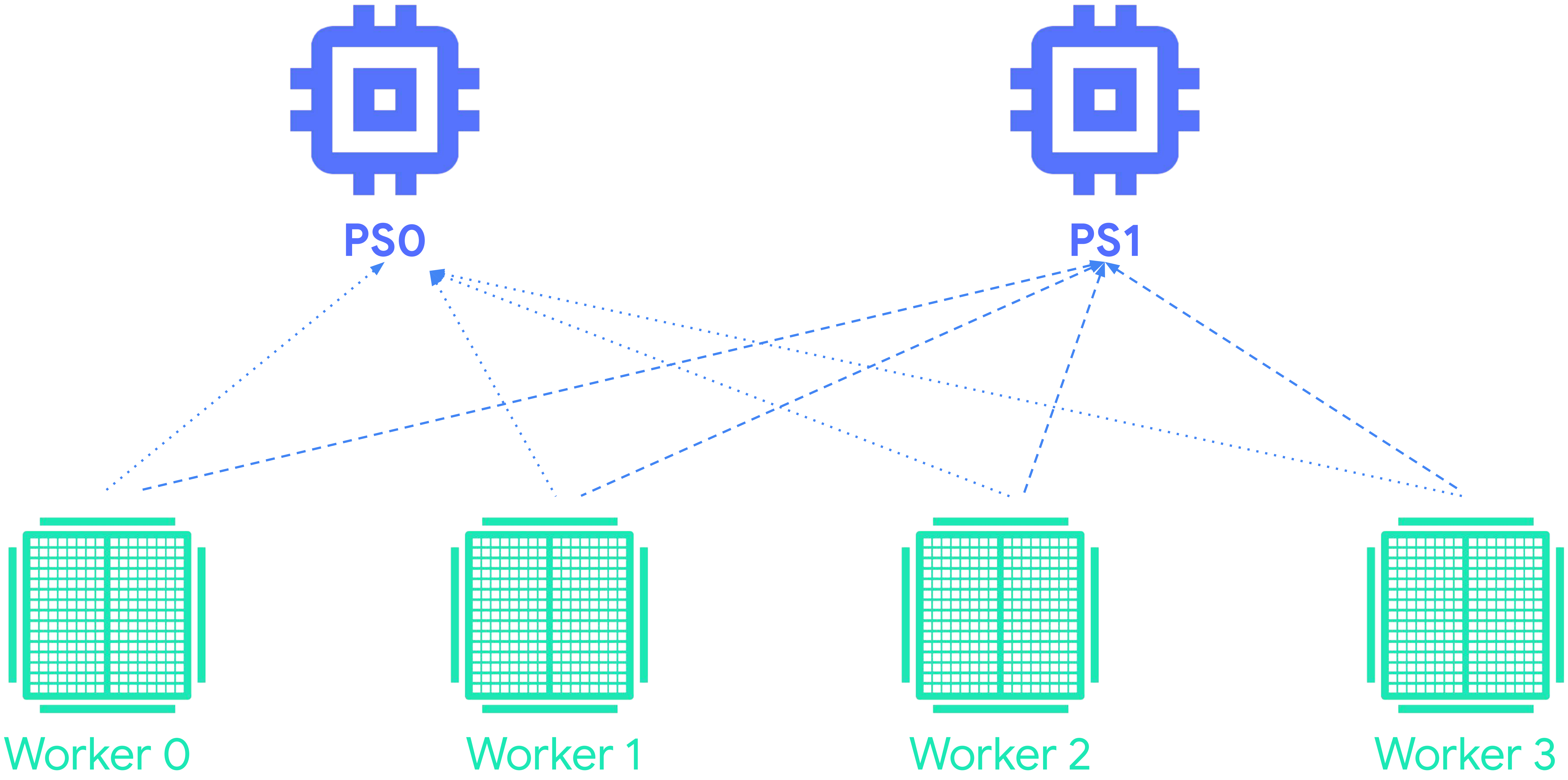
# Async Parameter Server



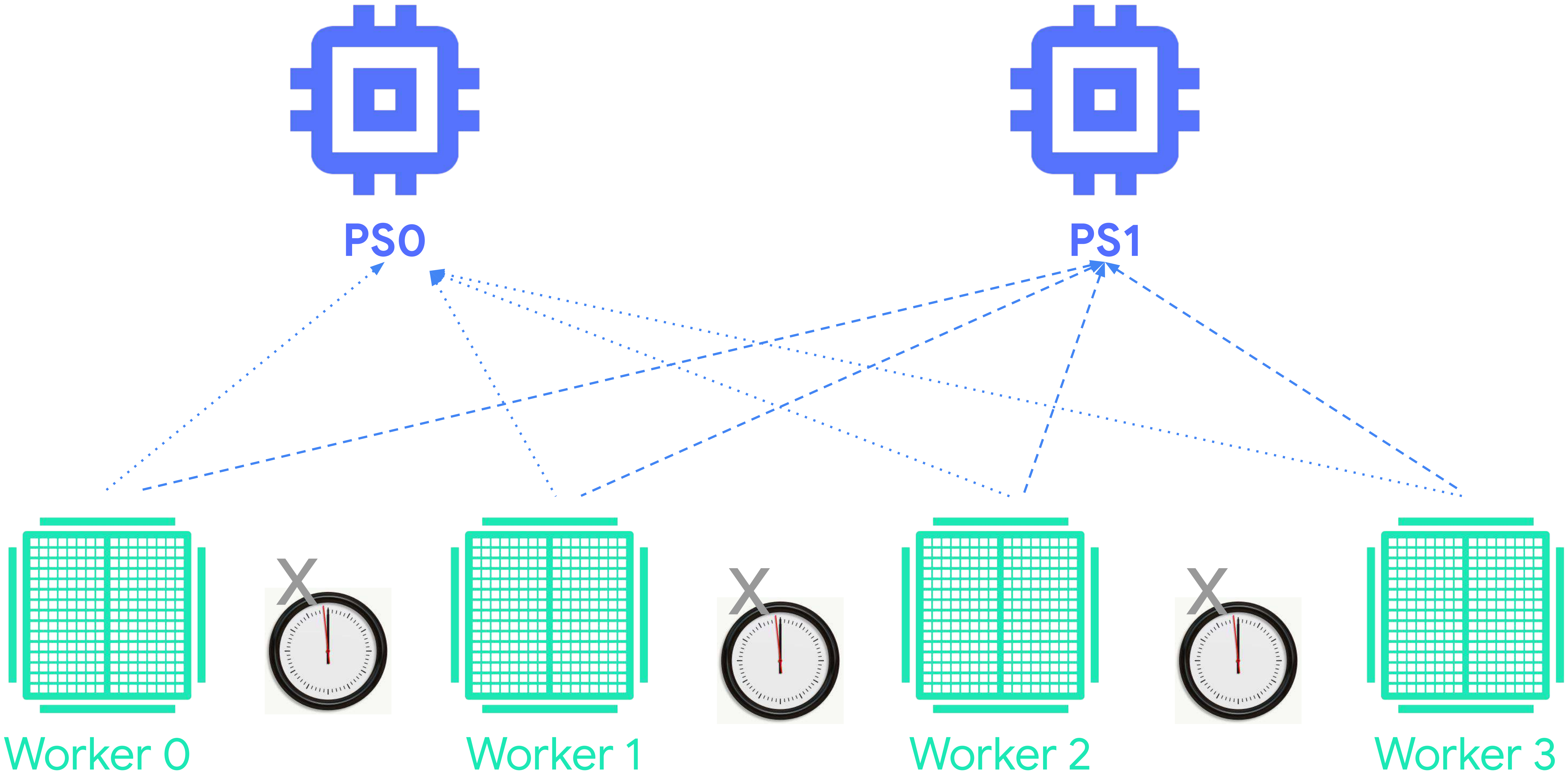
# Async Parameter Server



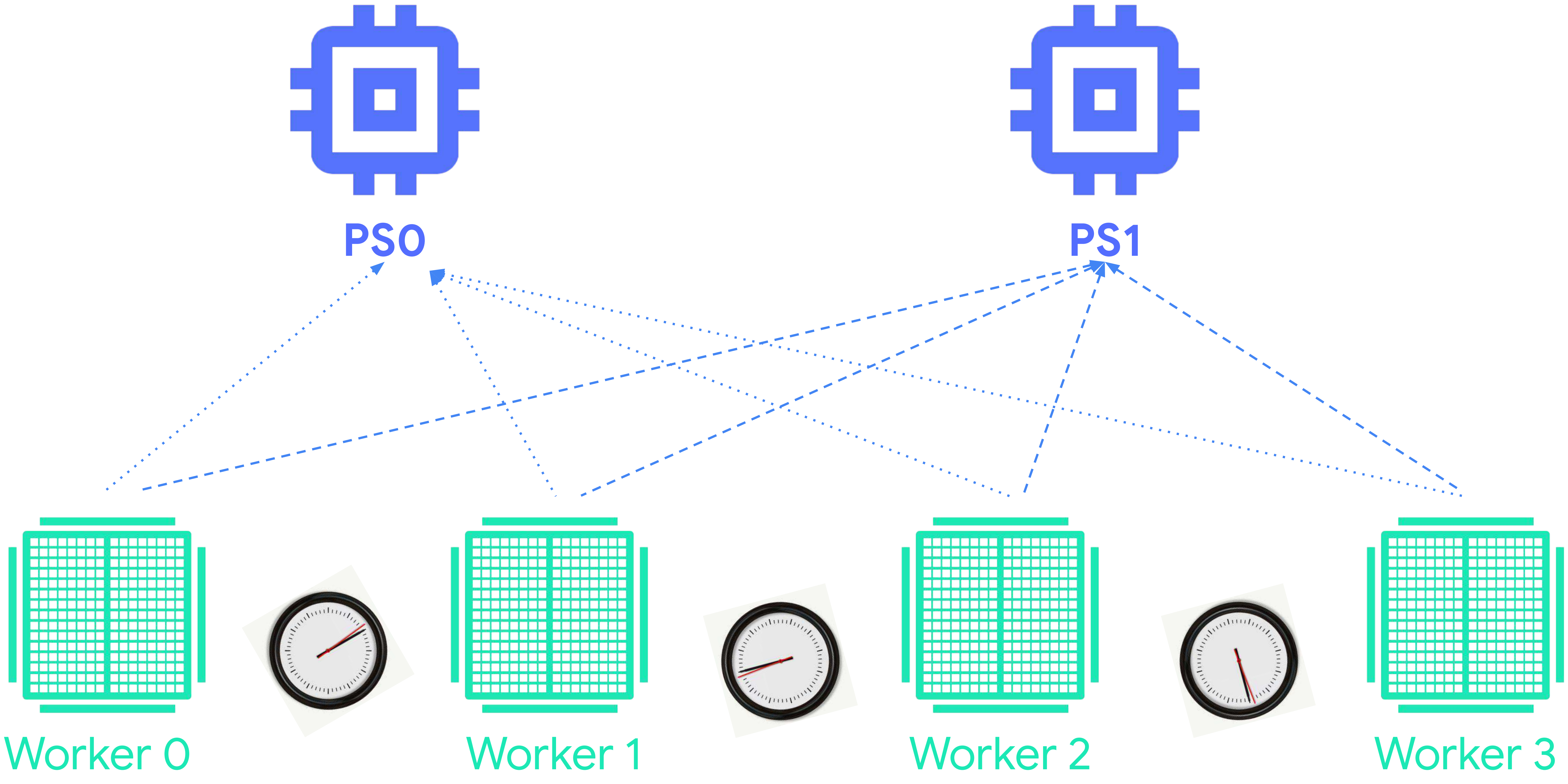
# Async Parameter Server



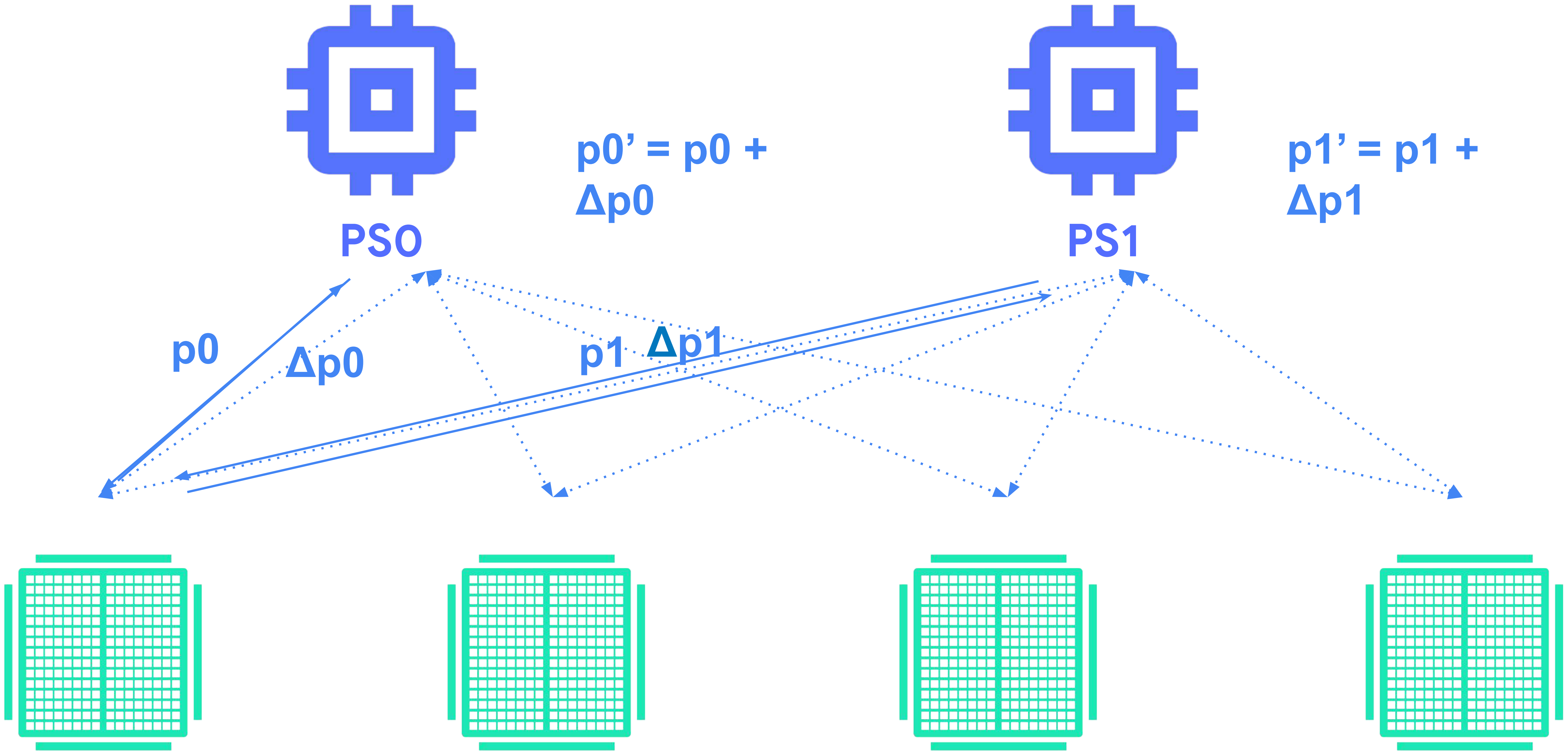
# Async Parameter Server

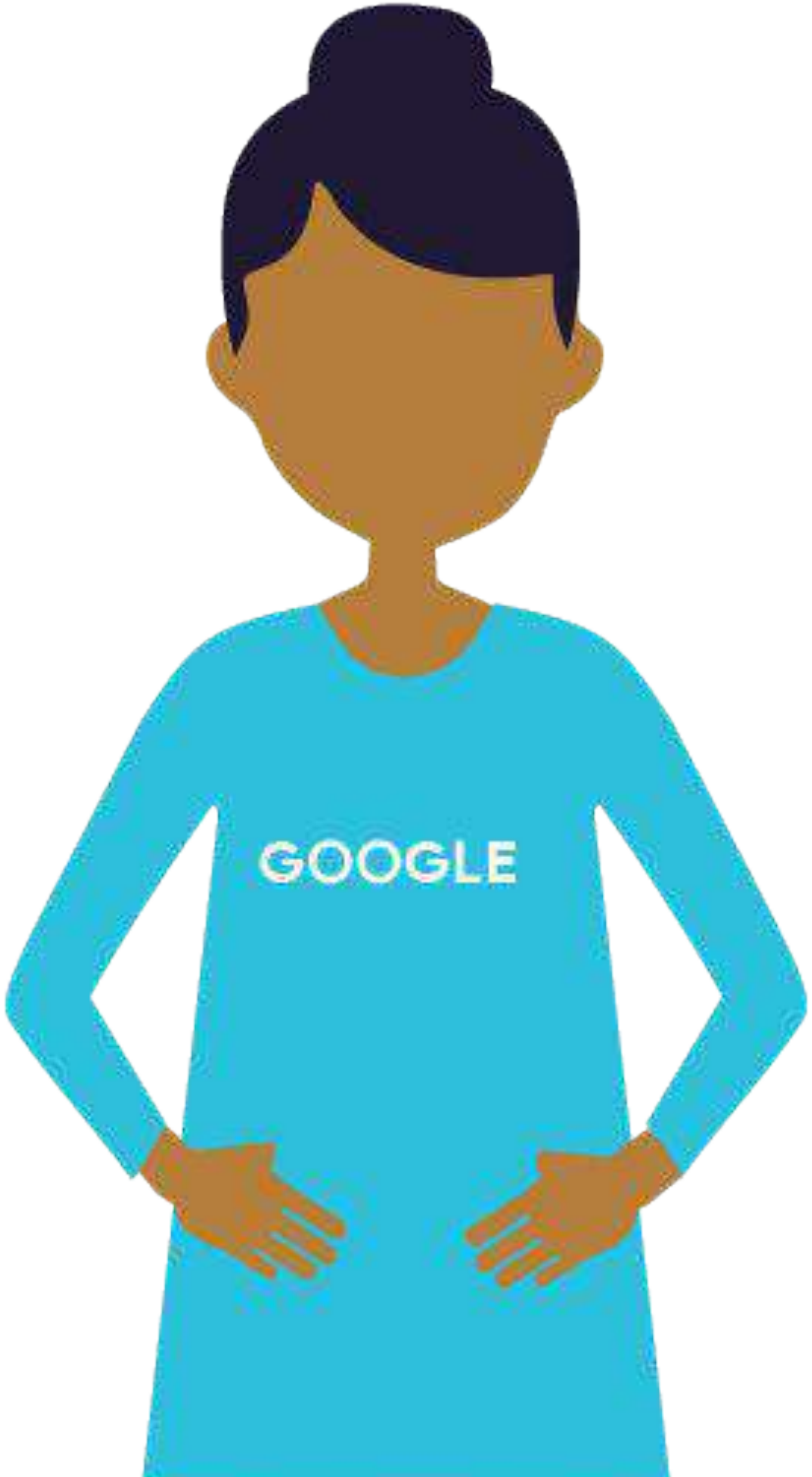


# Async Parameter Server



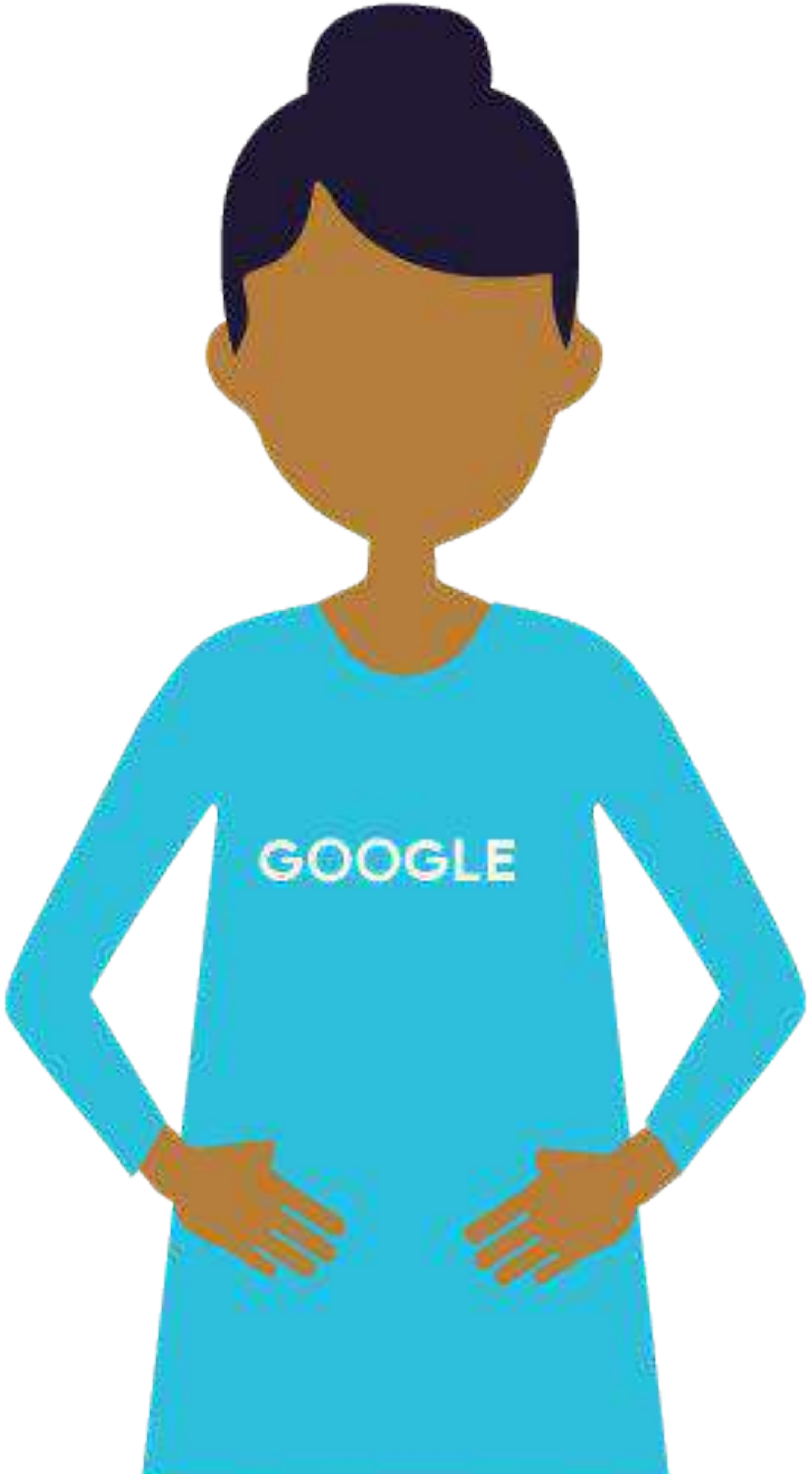
# Async Parameter Server



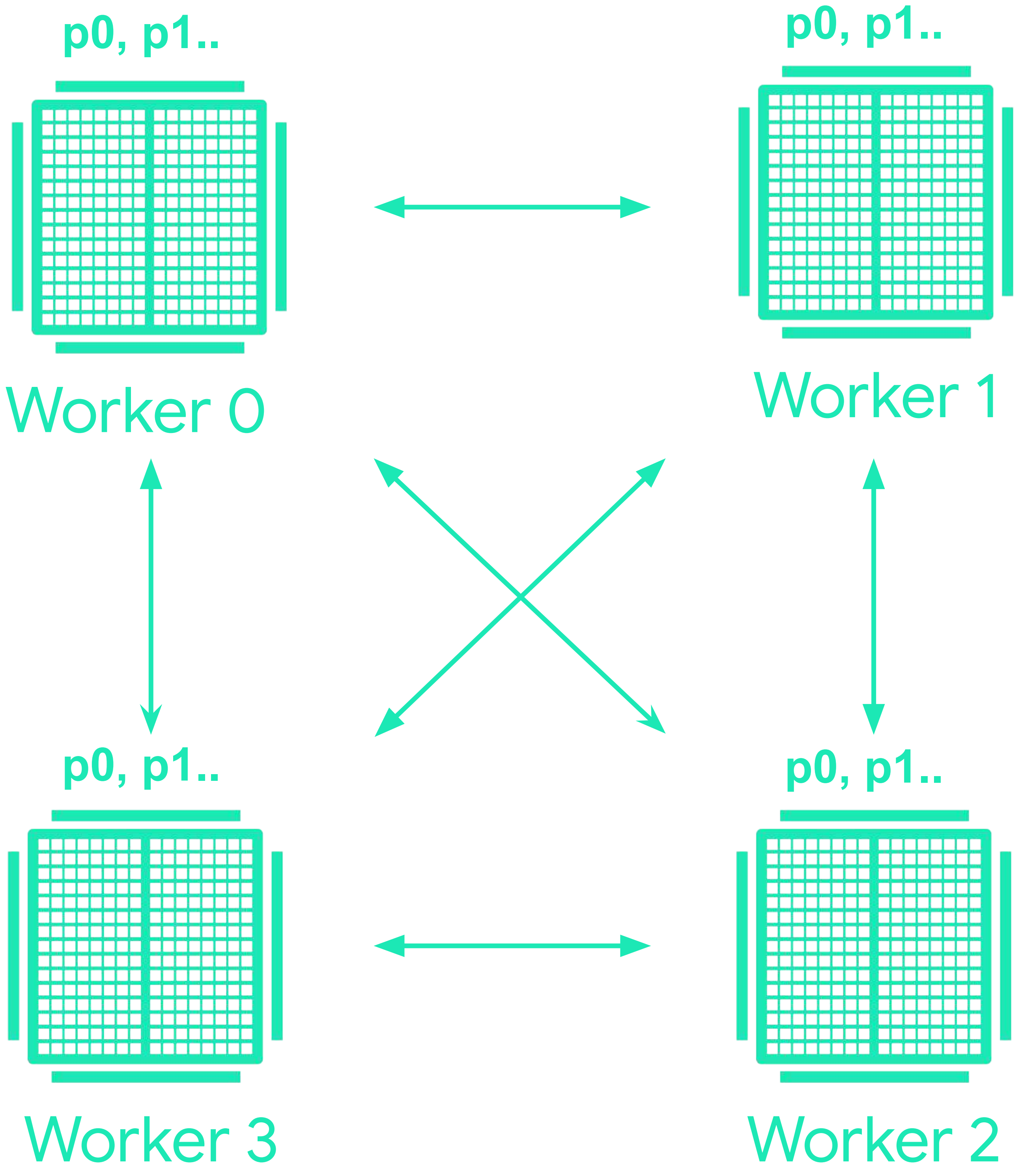


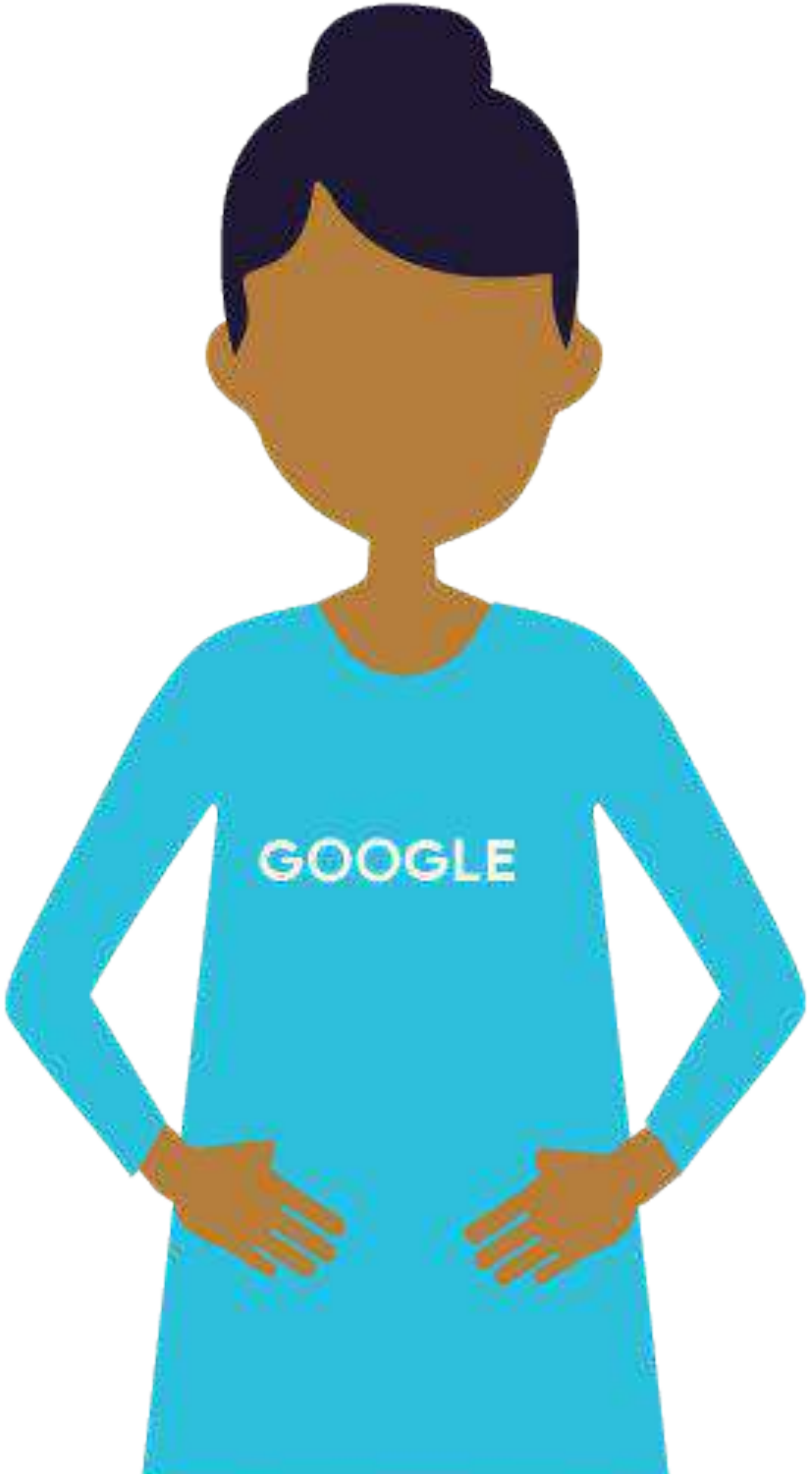

# Two approaches to Data Parallelism

1. Parameter server
2. **Sync Allreduce**

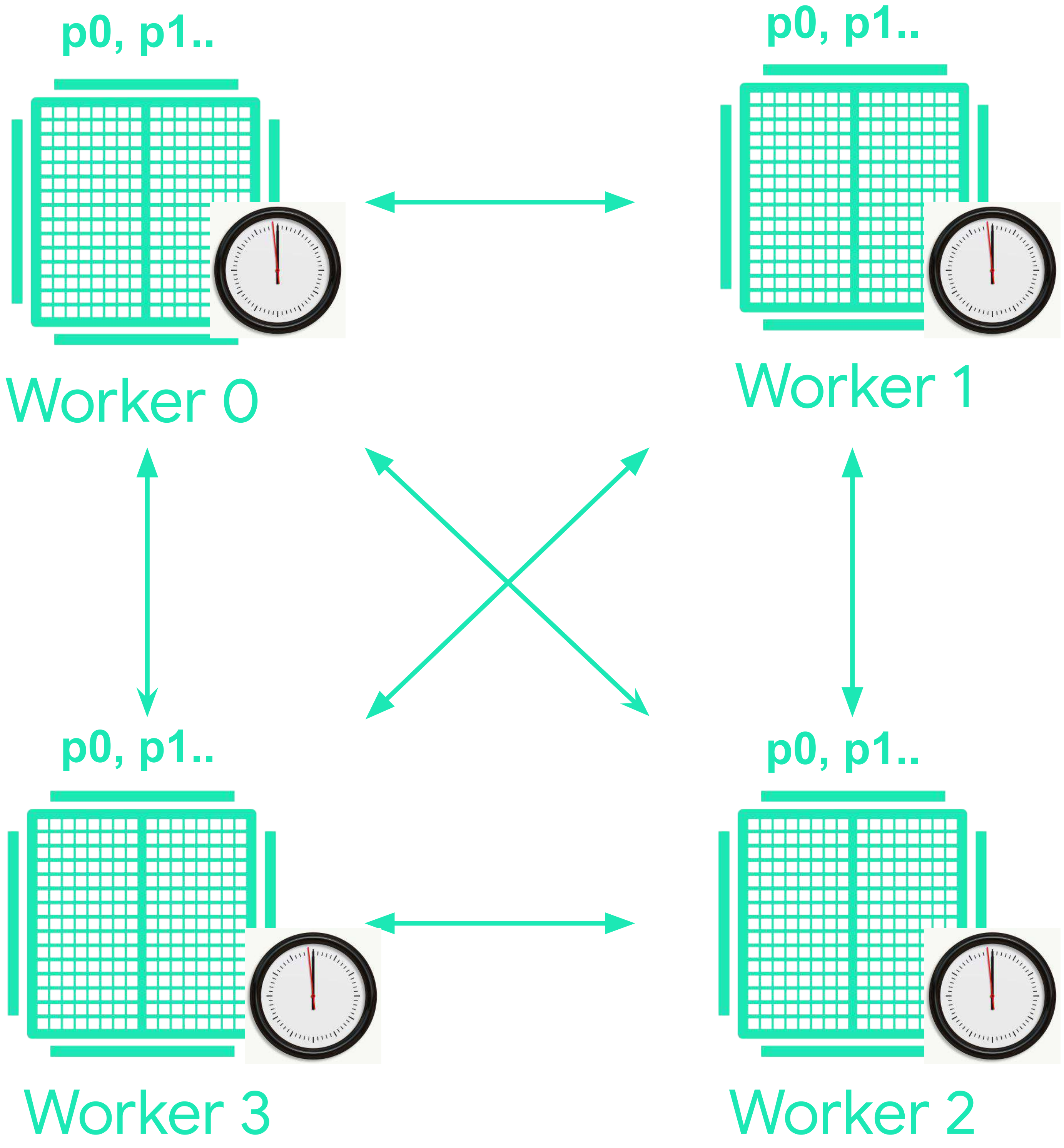


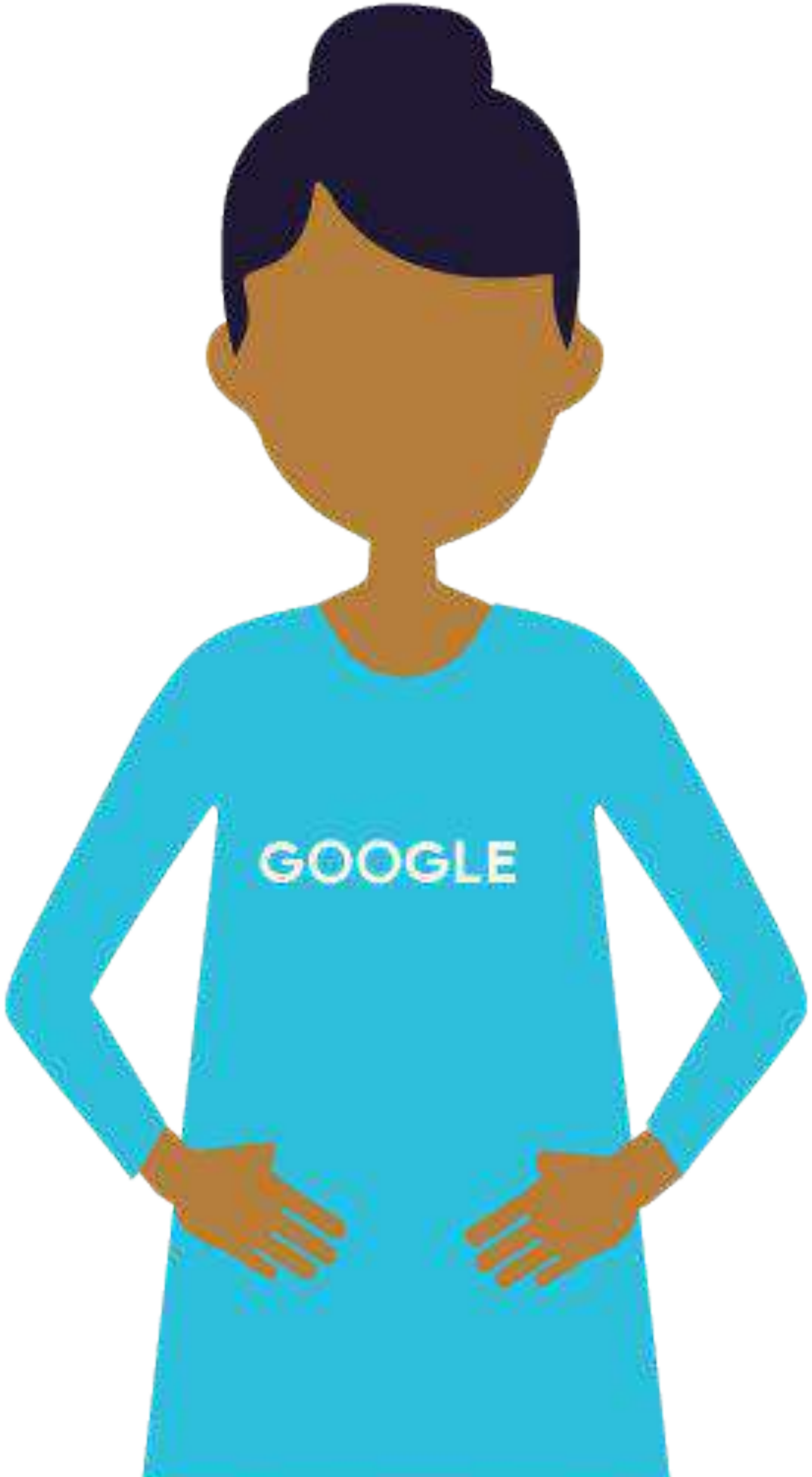

# Sync Allreduce Architecture



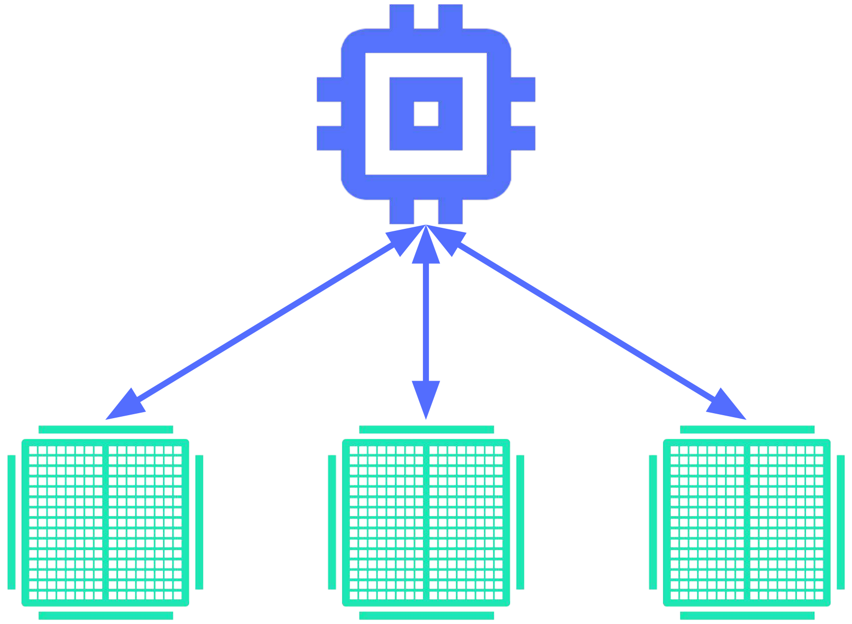



# Sync Allreduce Architecture

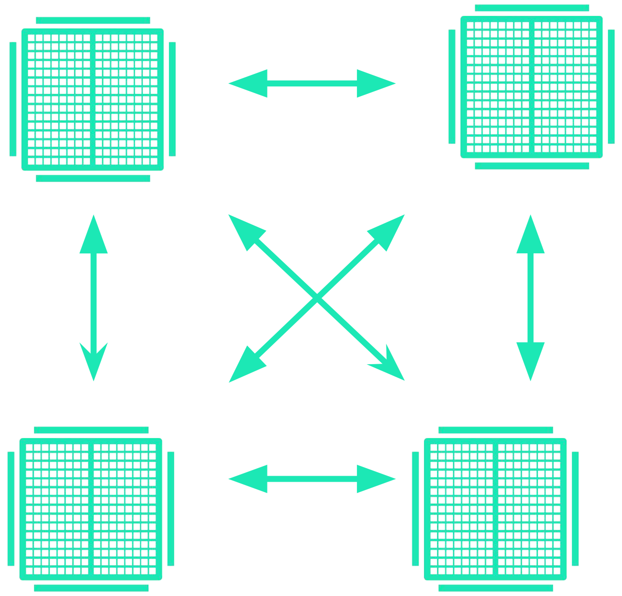




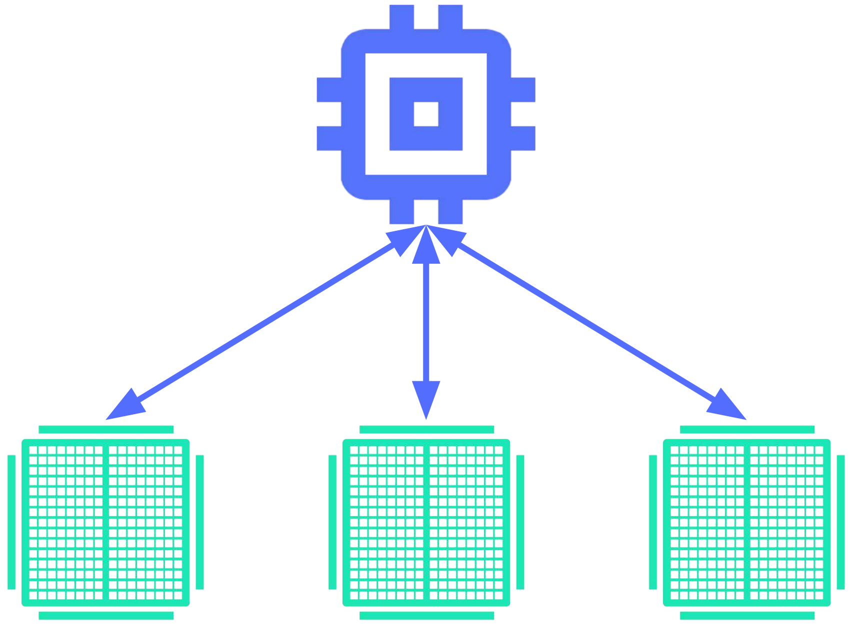

## Async Parameter Server



## Sync Allreduce

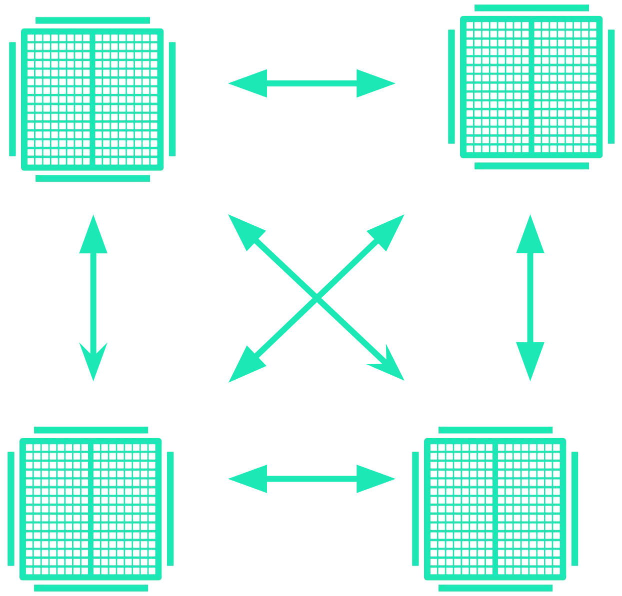


## Consider Async Parameter Server if...

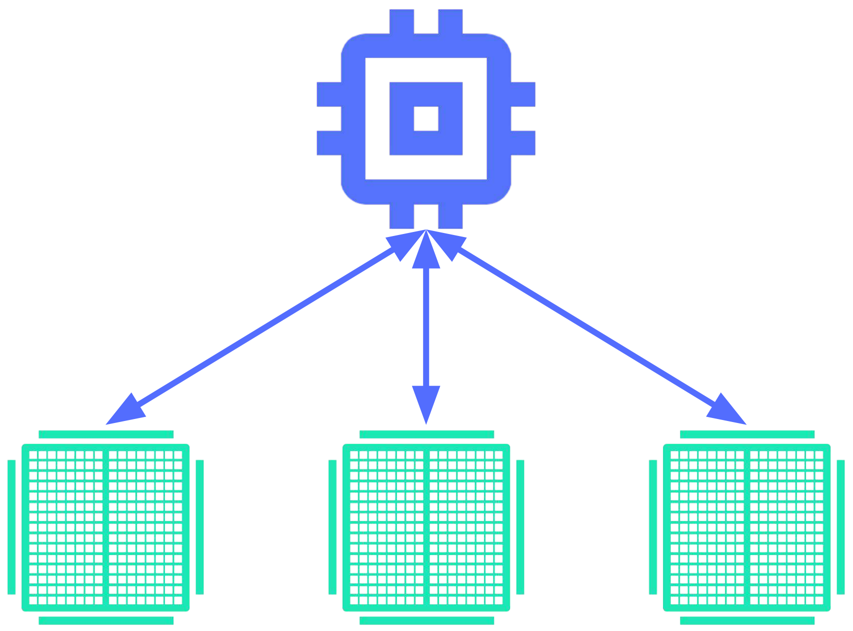


Many low-power or unreliable workers

## Consider Sync Allreduce if...

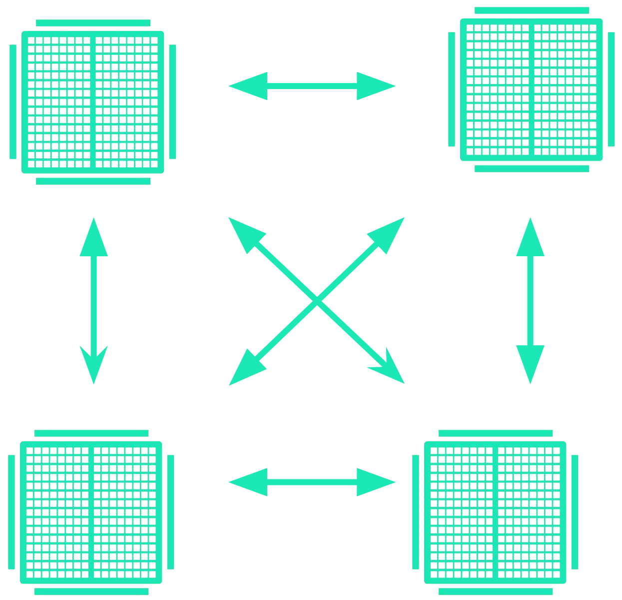


## Consider Async Parameter Server if...



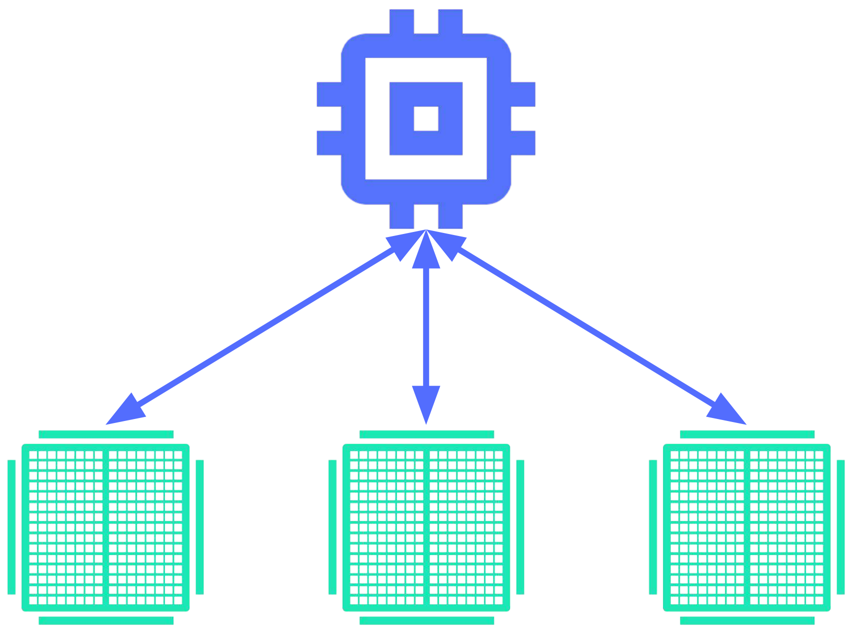
Many low-power
or unreliable workers

## Consider Sync Allreduce if...



Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

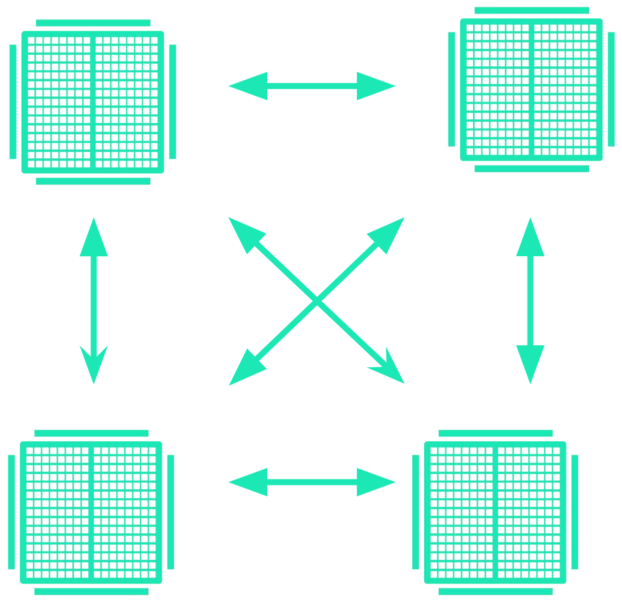
## Consider Async Parameter Server if...



Many low-power or unreliable workers

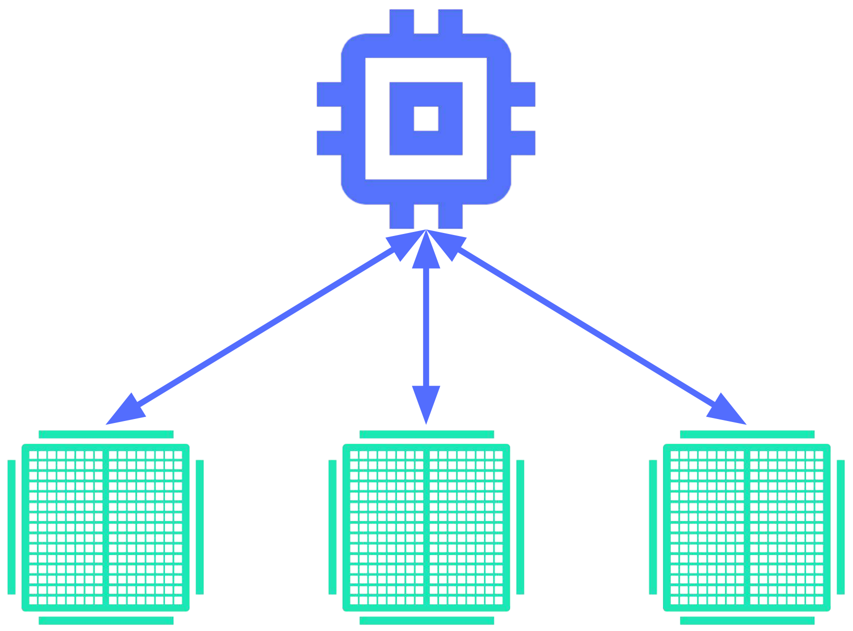
More mature approach

## Consider Sync Allreduce if...



Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

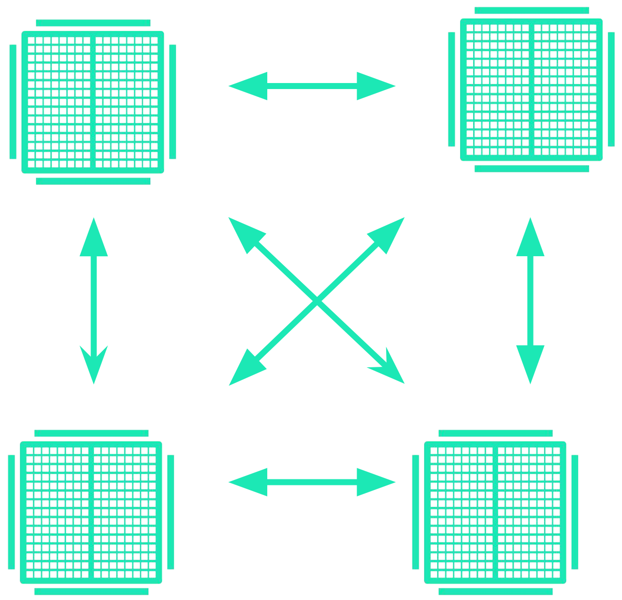
## Consider Async Parameter Server if...



Many low-power
or unreliable workers

More mature approach

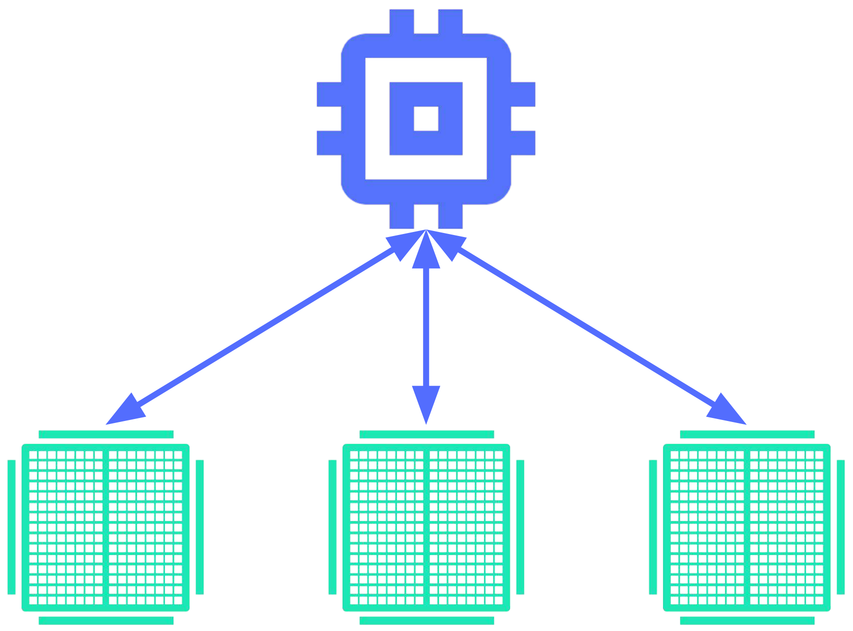
## Consider Sync Allreduce if...



Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

Better for multiple GPUs

## Consider Async Parameter Server if...

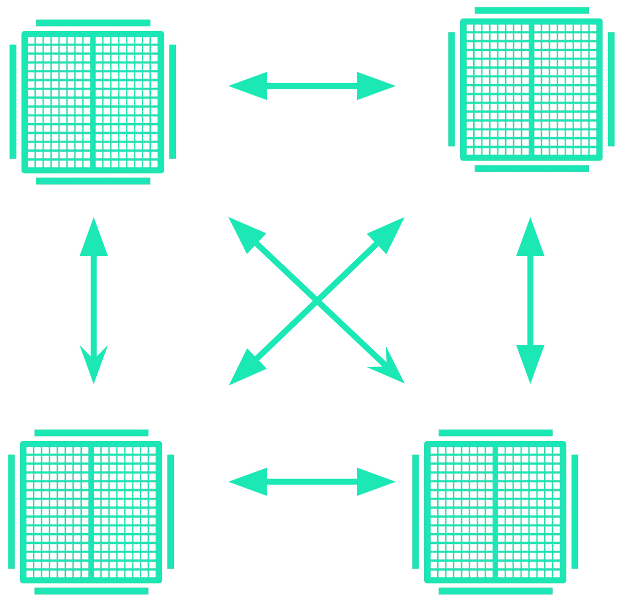


Many low-power or unreliable workers

More mature approach

Constrained by I/O

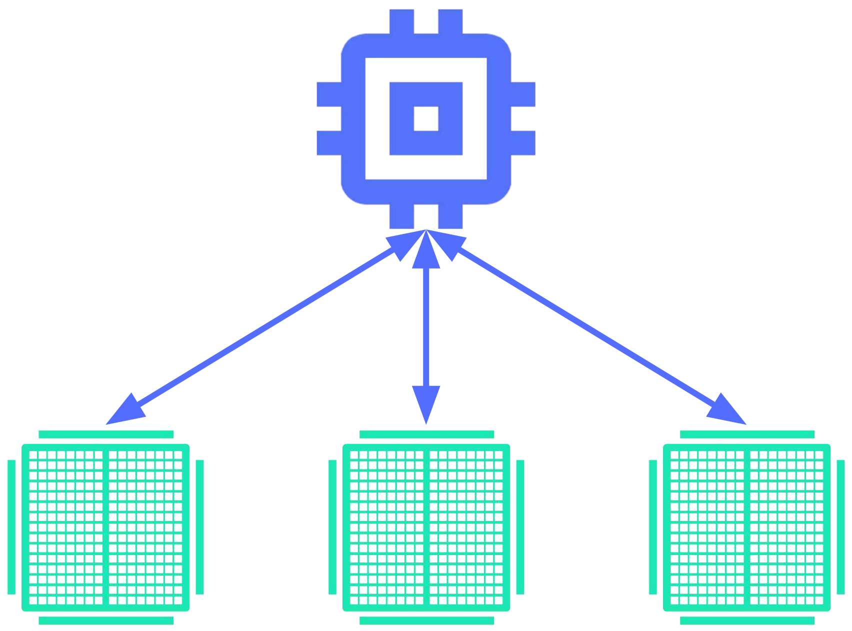
## Consider Sync Allreduce if...



Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

Better for multiple GPUs

## Consider Async Parameter Server if...

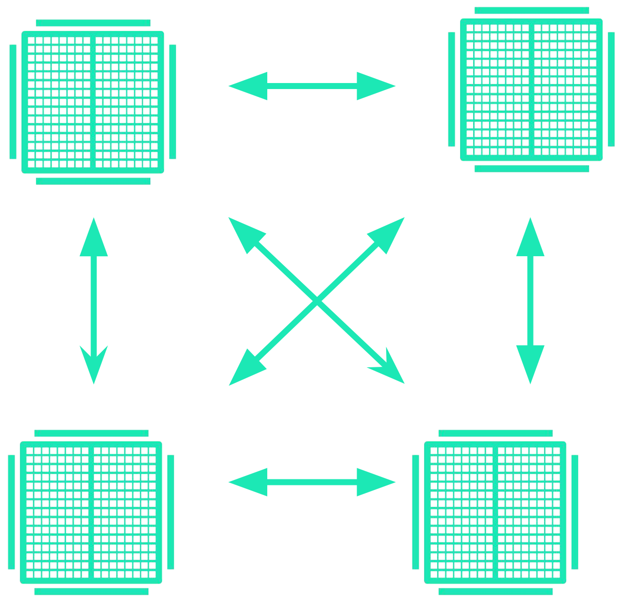


Many low-power
or unreliable workers

More mature approach

Constrained by I/O

## Consider Sync Allreduce if...

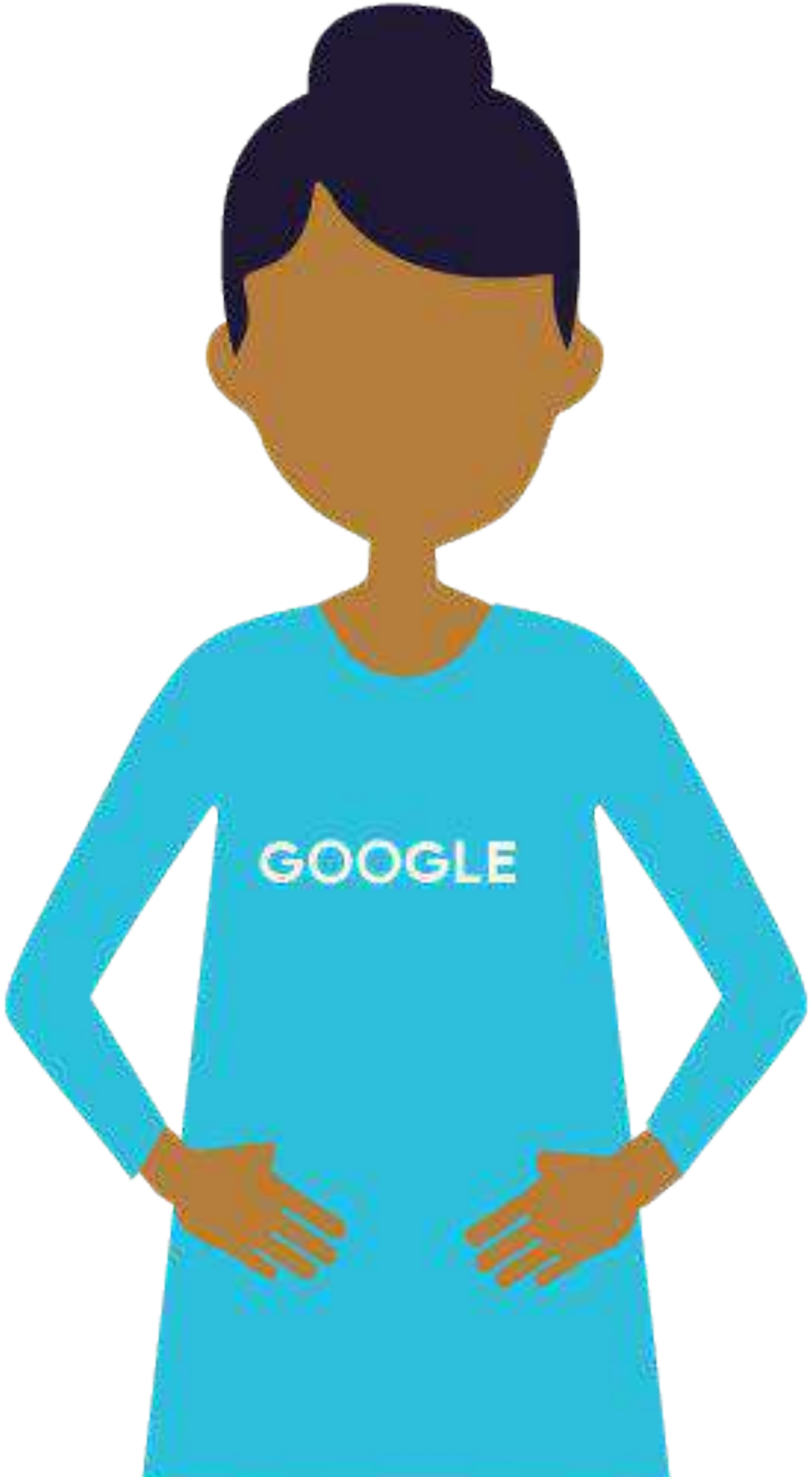


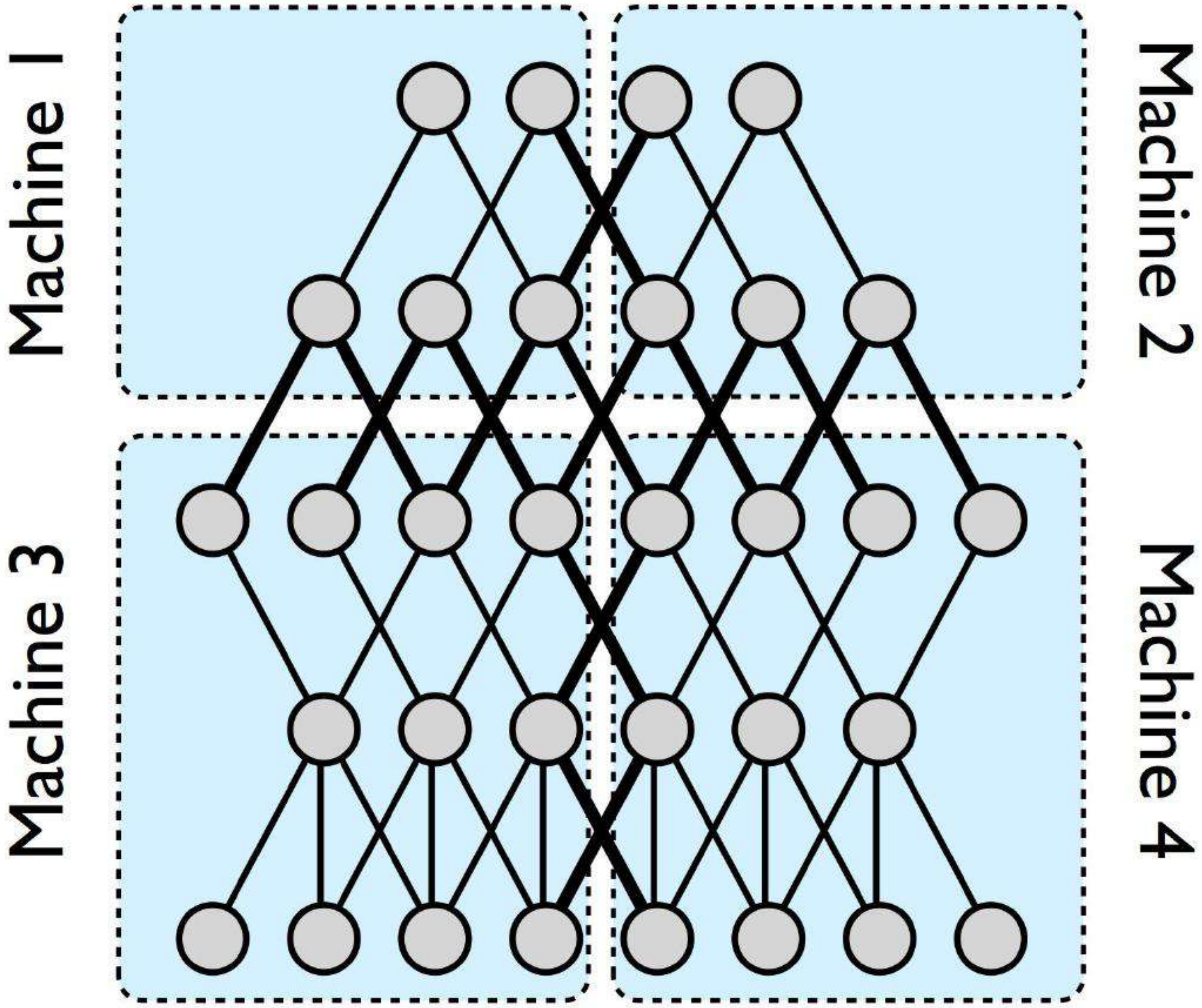
Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

Better for multiple GPUs

Constrained by compute power

# Model Parallelism



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Faster input pipelines**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l6_faster_input_pipelines

# Agenda

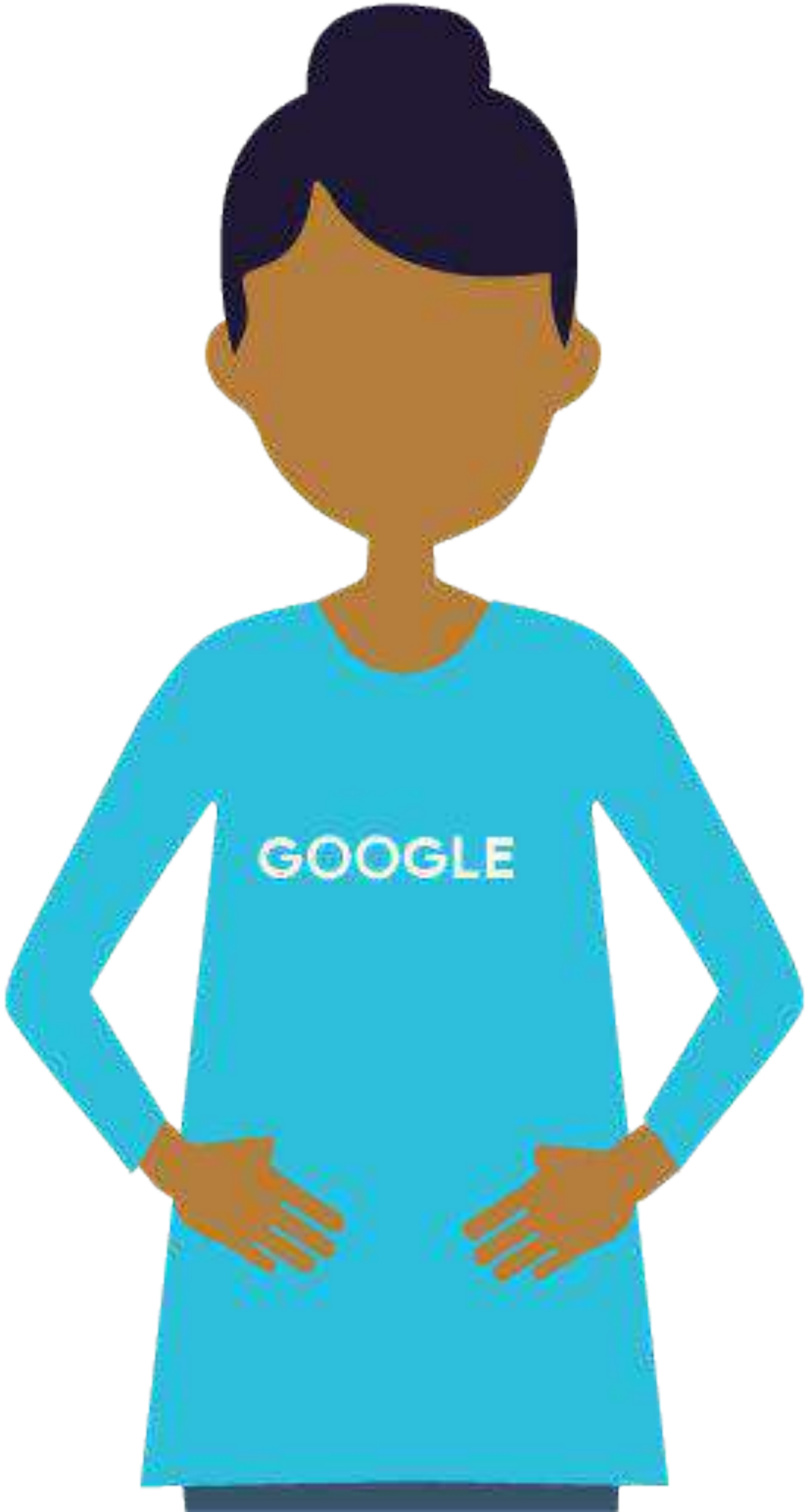
Distributed training

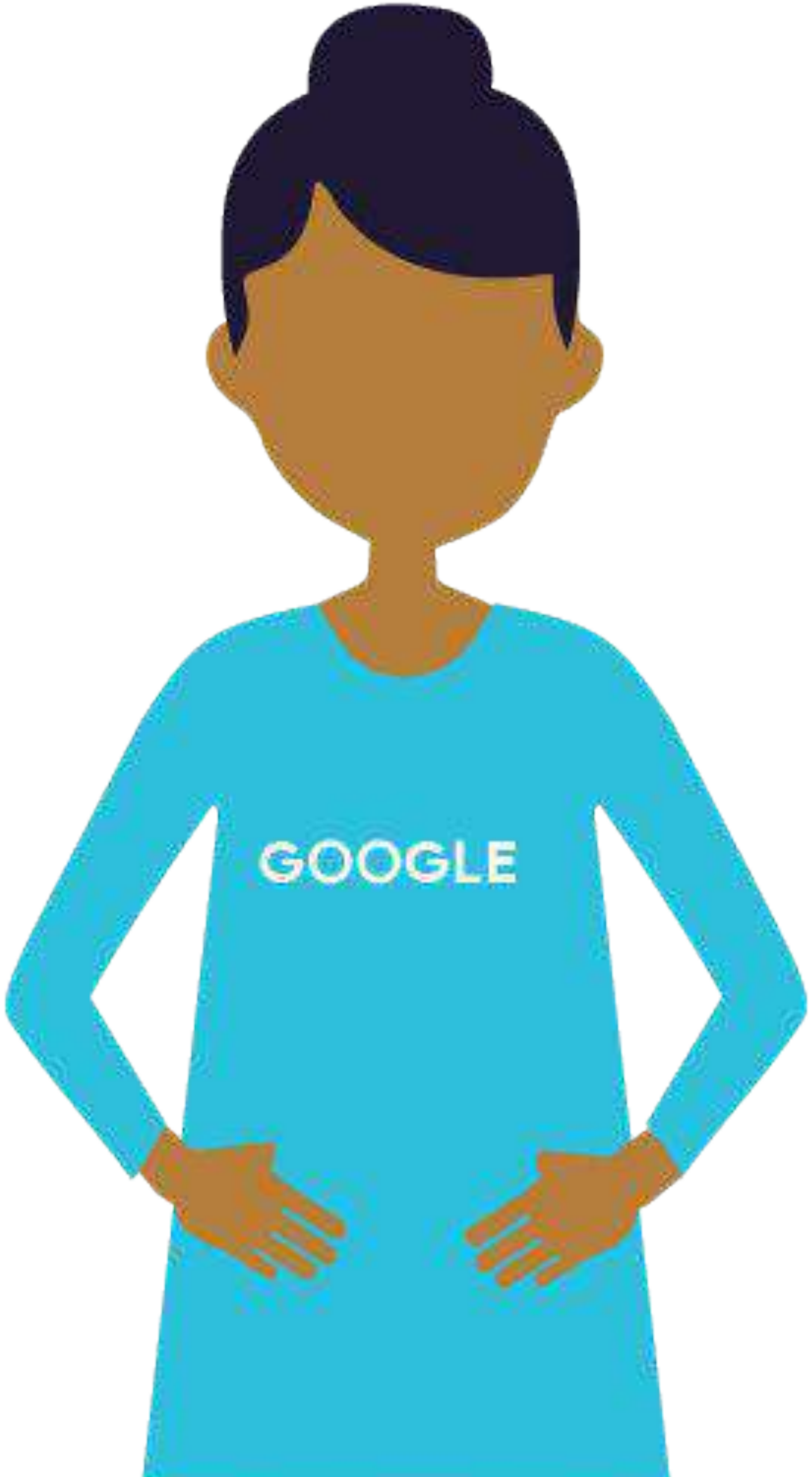
**Faster input pipelines**

Data parallelism (All Reduce)

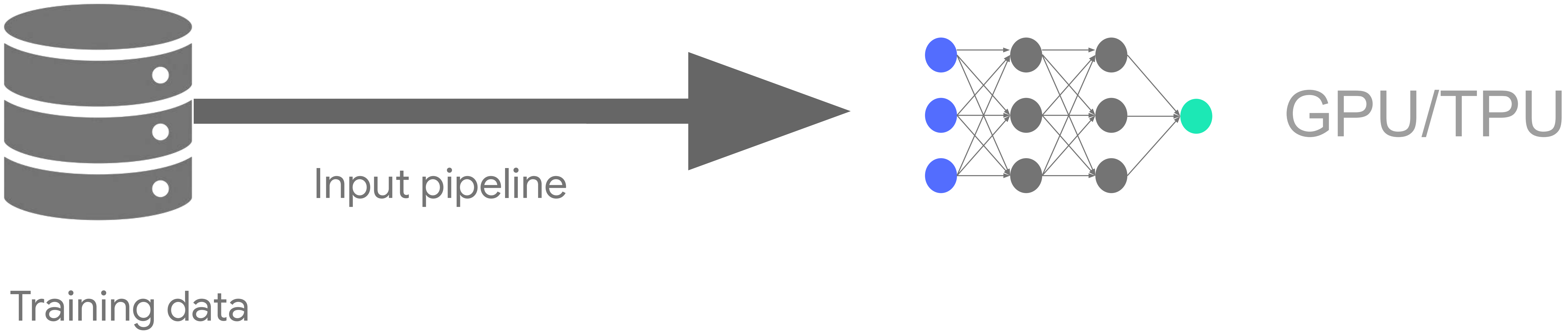
Parameter Server approach

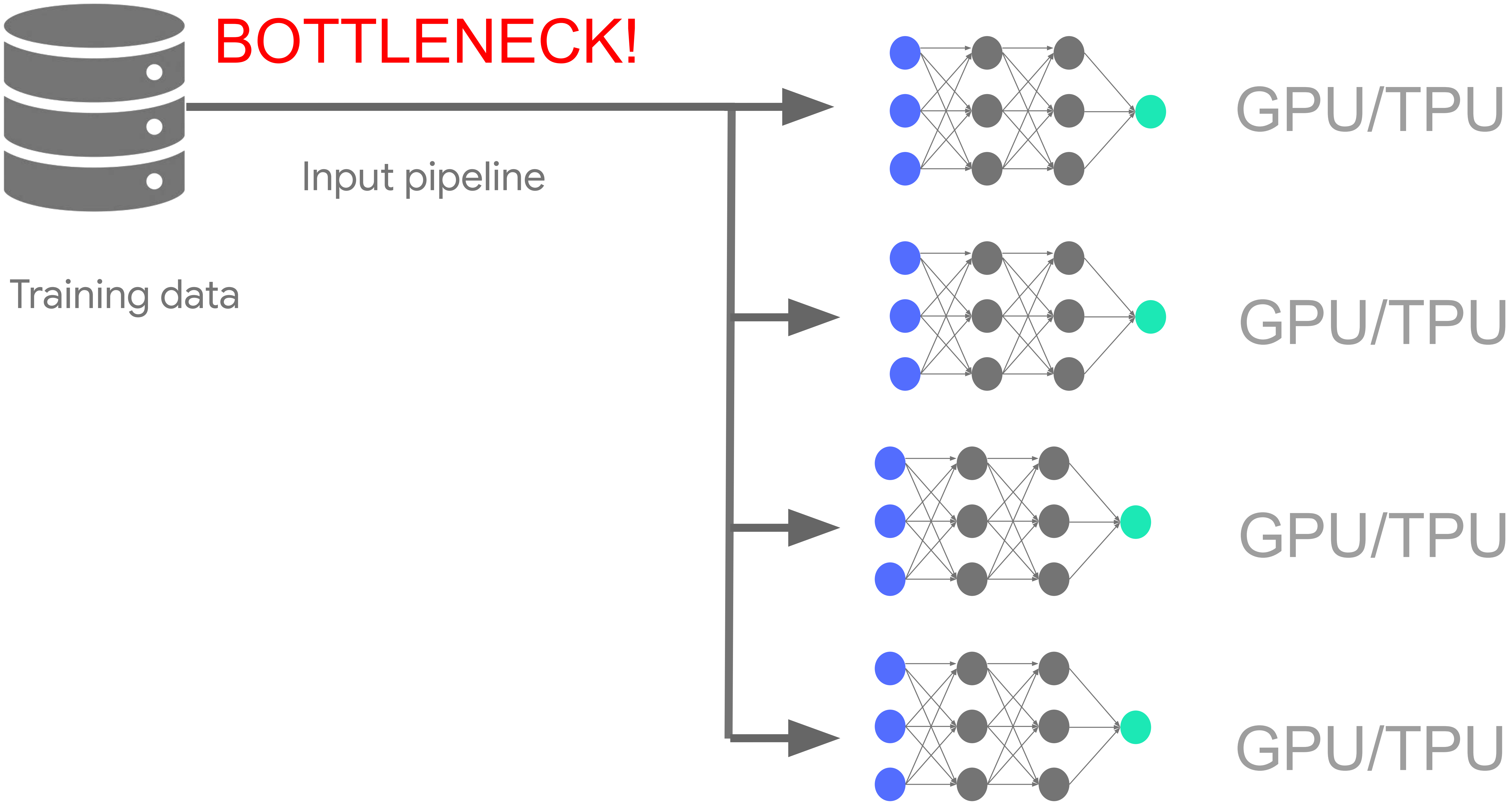
Inference

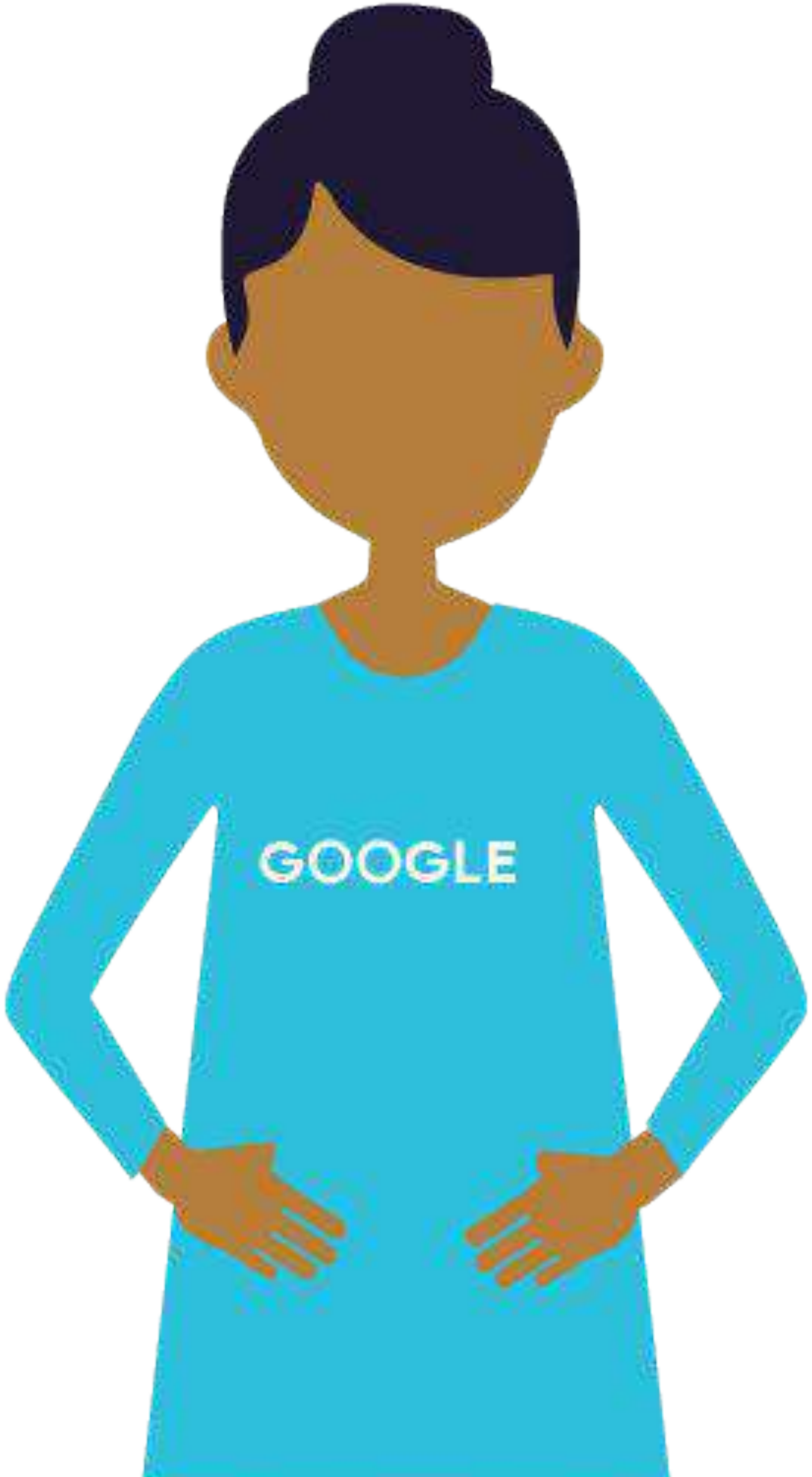





Faster input pipelines

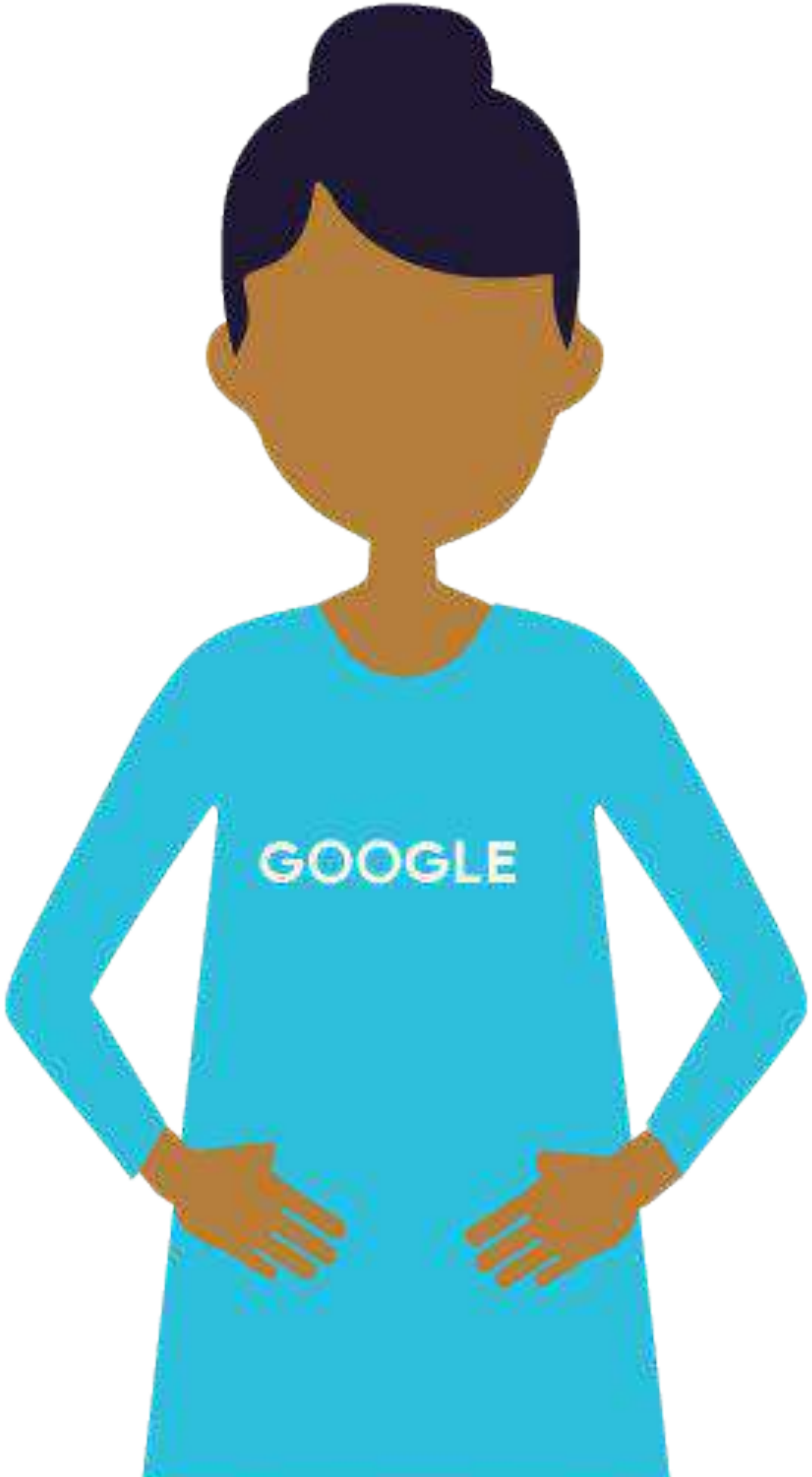

Input pipeline
GPU/TPU
Training data


BOTTLENECK!
Input pipeline
Training data
GPU/TPU
GPU/TPU
GPU/TPU
GPU/TPU




# Reading Data into TensorFlow

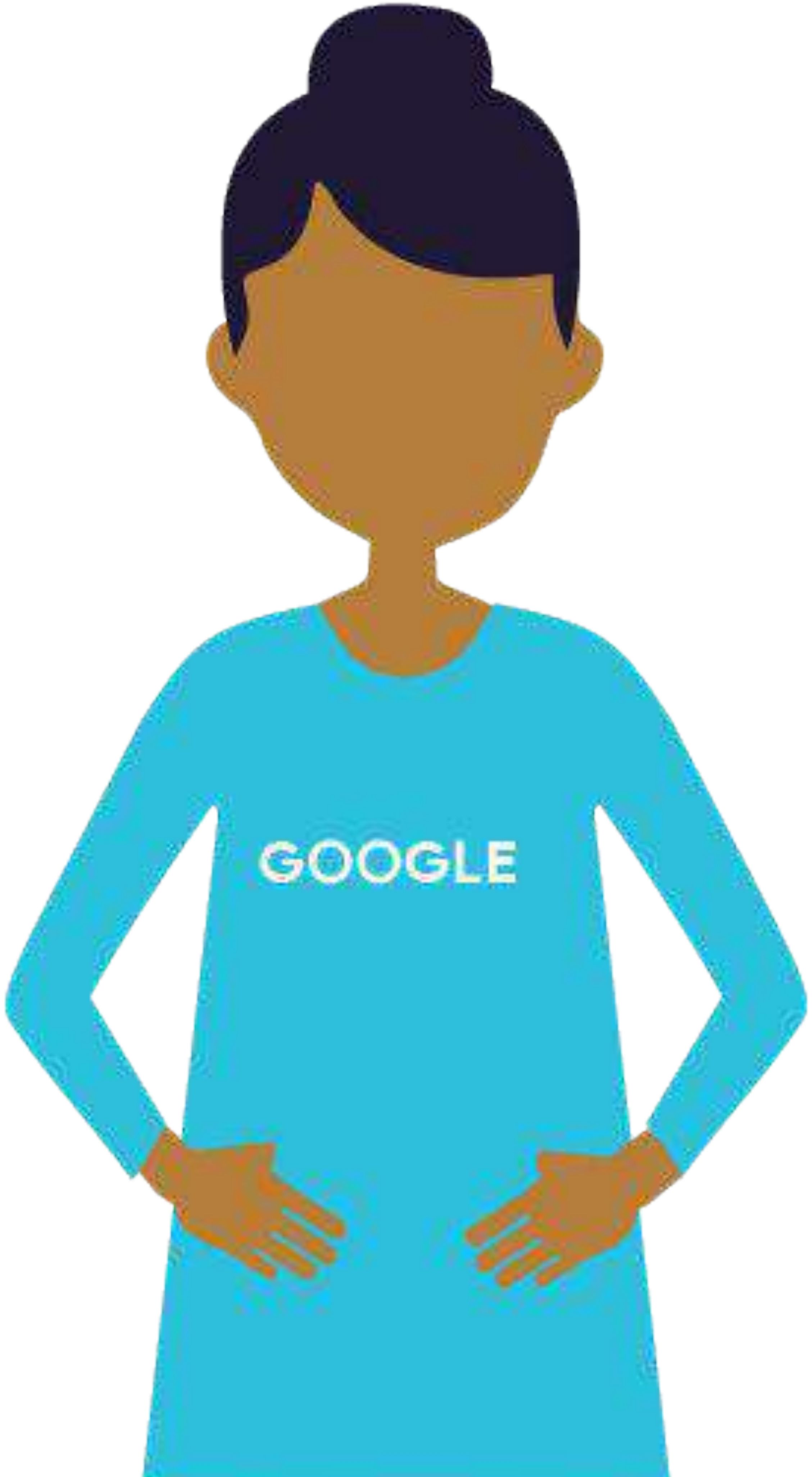
1. Directly feed from Python




Reading Data into TensorFlow

TensorFlow

2. Native TensorFlow Ops




# Reading Data into TensorFlow

3. Read transformed tf records

# 1. Feed TensorFlow directly from Python

```
def get_input_fn(data_set, num_epochs = None):
  return tf.estimator.inputs.pandas_input_fn(
      x = pd.DataFrame({k: data_set[k].values for k in FEATURES}),
      y = pd.Series(data_set[LABEL].values),
      num_epochs = num_epochs,
      shuffle = True, batch_size = 128)
```

Entire dataset in memory

Shuffle the data

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Native TensorFlow Operations**

Format: Screencast

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l7_native_tensorflow_operations

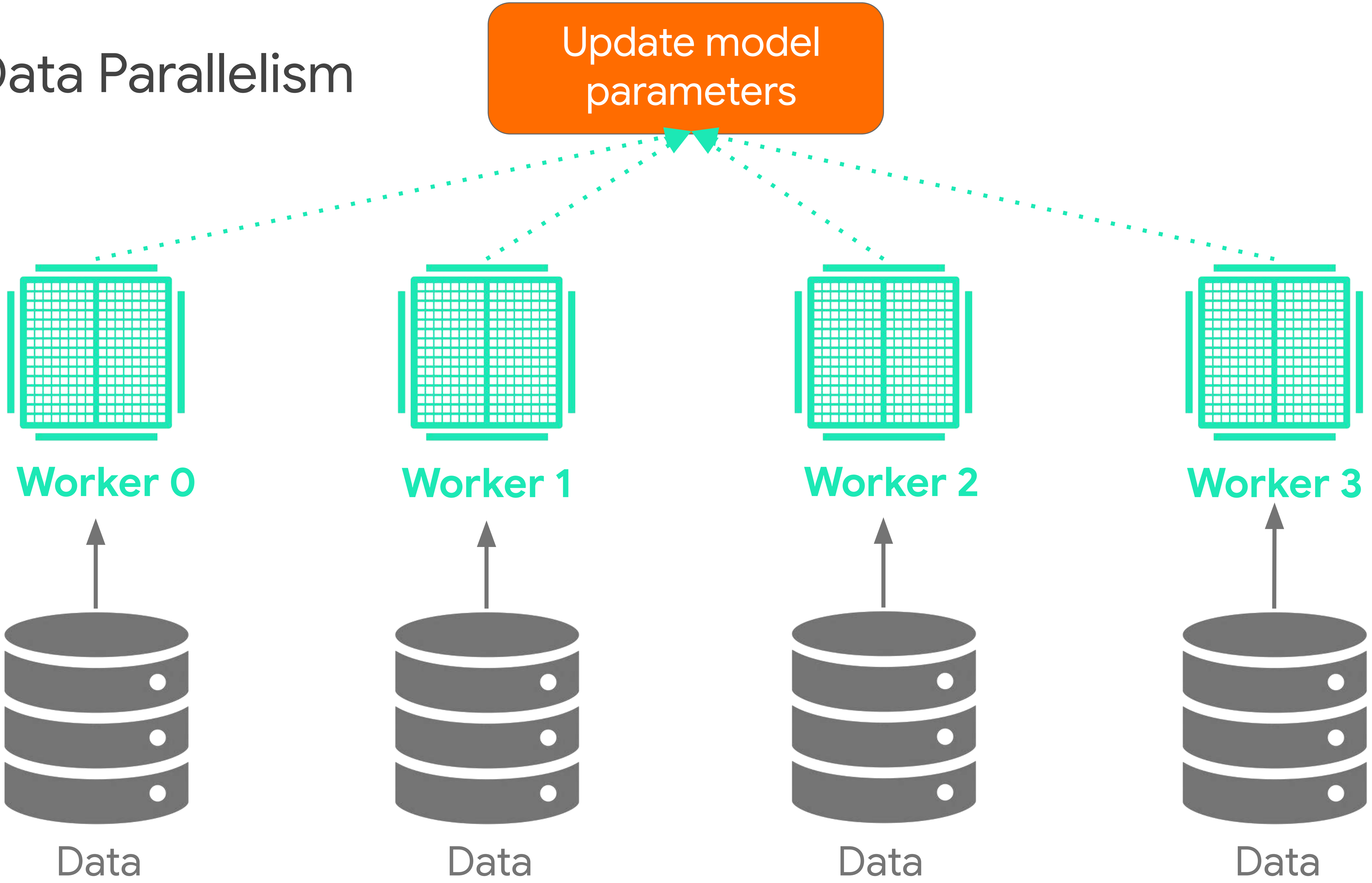
# 2. Using native TensorFlow ops to read CSV files

```
CSV_COLUMNS = ['fare_amount', 'pickuplon','pickuplat',...., 'key']
LABEL_COLUMN = 'fare_amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0], ['nokey']]

def read_dataset(filename, mode, batch_size = 512):
  def _input_fn():
    def decode_csv(value_column):
      columns = tf.decode_csv(value_column, record_defaults = DEFAULTS)
      features = dict(zip(CSV_COLUMNS, columns))
      label = features.pop(LABEL_COLUMN)
      return features, label

    file_list = tf.gfile.Glob(filename) # create list of files that match pattern
    dataset = tf.data.TextLineDataset(file_list).map(decode_csv) # create dataset from file list
    if mode == tf.estimator.ModeKeys.TRAIN:
        num_epochs = None # indefinitely
        dataset = dataset.shuffle(buffer_size = 10 * batch_size)
    else:
        num_epochs = 1 # end-of-input after this

    dataset = dataset.repeat(num_epochs).batch(batch_size)
    return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

# 2. Using native TensorFlow ops to read CSV files

```
CSV_COLUMNS = ['fare_amount', 'pickuplon','pickuplat',..., 'key']
LABEL_COLUMN = 'fare_amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0], ['nokey']]

def read_dataset(filename, mode, batch_size = 512):
  def _input_fn():
    def decode_csv(value_column):
      columns = tf.decode_csv(value_column, record_defaults = DEFAULTS)
      features = dict(zip(CSV_COLUMNS, columns))
      label = features.pop(LABEL_COLUMN)
      return features, label

    file_list = tf.gfile.Glob(filename) # create list of files that match pattern
    dataset = tf.data.TextLineDataset(file_list).map(decode_csv) # create dataset from file list
    if mode == tf.estimator.ModeKeys.TRAIN:
        num_epochs = None # indefinitely
        dataset = dataset.shuffle(buffer_size = 10 * batch_size)
    else:
        num_epochs = 1 # end-of-input after this

    dataset = dataset.repeat(num_epochs).batch(batch_size)
    return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

# Data Parallelism



# 2. Using native TensorFlow ops to read images

```
# Reads an image from a file, decodes it into a dense tensor, and resizes it to a fixed shape.
def _parse_function(filename, label):
  image_string = tf.read_file(filename)
  image_decoded = tf.image.decode_image(image_string)
  image_resized = tf.image.resize_images(image_decoded, [299, 299])
  return image_resized, label

# A vector of filenames.
file_list = tf.gfile.Glob(filename)
filenames = tf.constant(file_list)

# labels[i] is the label for the image in filenames[i].
labels = tf.constant(label_list)
dataset = tf.data.Dataset.from_tensor_slices((filenames, labels))
dataset = dataset.map(_parse_function)
```

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **TensorFlow Records**

Format: Screencast

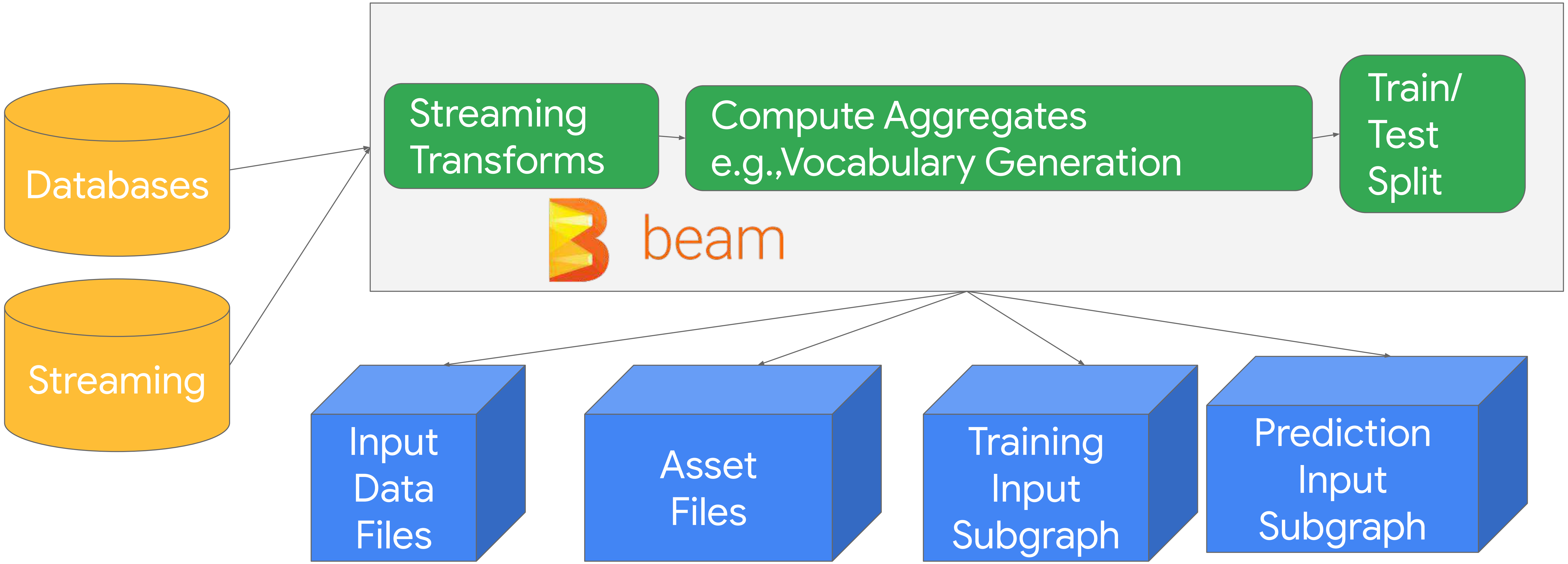
Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l8_tensorflow_records

# 3. Preprocess data into TFRecord

```
def convert_to_example(csvline, categories):
  filename, label = csvline.encode('ascii', 'ignore').split(',')
  if label in categories:
    coder = ImageCoder()
    image_buffer, height, width = _get_image_data(filename, coder)
    example = _convert_to_example(filename, image_buffer,
                                  categories.index(label), label, height, width)
    yield example.SerializeToString()

LABELS = ['nails', 'screws']
(p
 | beam.FlatMap(lambda line: convert_to_example(line, LABELS))
 | beam.io.tfrecordio.WriteToTFRecord(os.path.join(OUTPUT_DIR, 'train')))
)

# https://github.com/tensorflow/tpu/blob/master/tools/datasets/jpeg_to_tf_record.py
```

# 3. Can use Tensorflow Transform to create tf records



# 3. Writing TFRecord from Spark


```
import org.apache.spark.sql.DataFrame


df.write.format("tfrecords").option("recordType", "Example").save(path)
```

Warning! Because preprocessing is carried out on DataFrame using Spark, you need to repeat the preprocessing during prediction using Spark Streaming (the code is different).

# 3. Read TFRecord produced by tf.transform or Spark

```
from tensorflow_transform.saved import input_fn_maker
def gzip_reader_fn():
  return tf.TFRecordReader(options=tf.python_io.TFRecordOptions(
      compression_type=tf.python_io.TFRecordCompressionType.GZIP))

def get_input_fn(transformed_metadata, transformed_data_paths, batch_size, mode):
  return input_fn_maker.build_training_input_fn(
      metadata=transformed_metadata,
      file_pattern=(
          transformed_data_paths[0] if len(transformed_data_paths) == 1
          else transformed_data_paths),
      training_batch_size=batch_size,
      label_keys=[TARGET_FEATURE_COLUMN],
      reader=gzip_reader_fn,
      key_feature_name=KEY_FEATURE_COLUMN,
      reader_num_threads=4,
      queue_capacity=batch_size * 2,
      randomize_input=(mode != tf.contrib.learn.ModeKeys.EVAL),
      num_epochs=(1 if mode == tf.contrib.learn.ModeKeys.EVAL else None))
```

gzipped files

tf.transform writes out the TFRecords with metadata

Each read is of Batch_SIZE recs

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

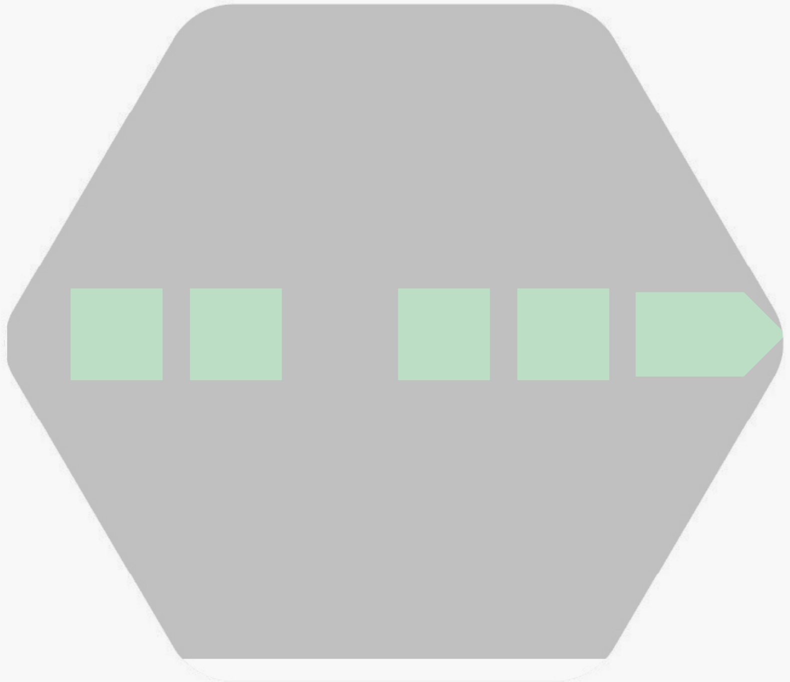
Lesson Title: **Parallel pipelines**

Format: Screencast

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l9_parallel_pipelines

# Three approaches to reading data into TensorFlow



Directly feed from Python

Native TensorFlow ops

Read transformed tf records

## A simple input pipeline for an image model

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files)
  dataset = dataset.shuffle(10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn)
  dataset = dataset.batch(batch_size)
  return dataset
```

# Input pipeline as an ETL Process

## A simple input pipeline for an image model

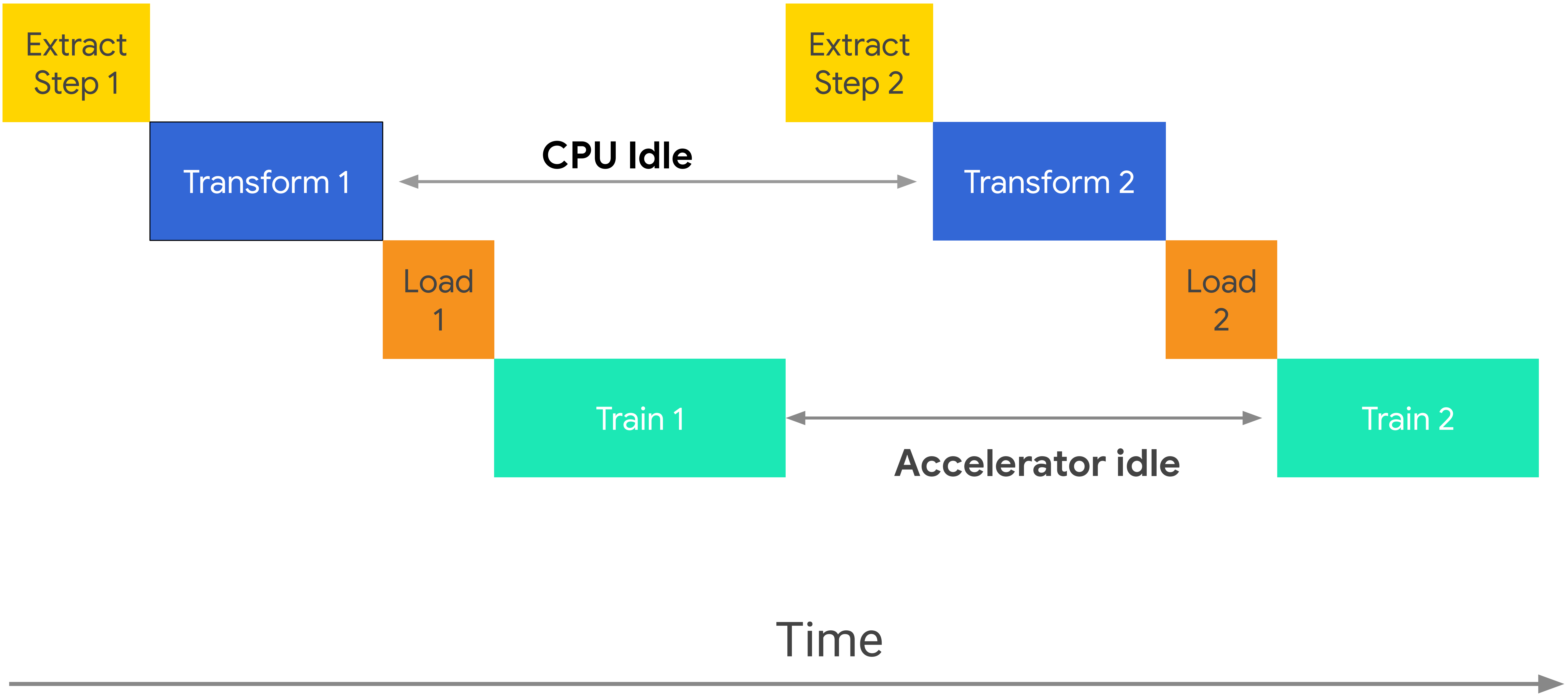
E

T

L

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files)

  dataset = dataset.shuffle(10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn)
  dataset = dataset.batch(batch_size)

  return dataset
```

# Input pipeline bottleneck



# # 1. Parallelize file reading

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
  dataset = dataset.shuffle(buffer_size=10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn)
  dataset = dataset.batch(batch_size)
  return dataset
```

Parallelize
file reading
from Google
Cloud Storage

# # 2. Parallelize map for transformations

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
  dataset = dataset.shuffle(buffer_size=10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn, num_parallel_calls=40)
  dataset = dataset.batch(batch_size)
  return dataset
```

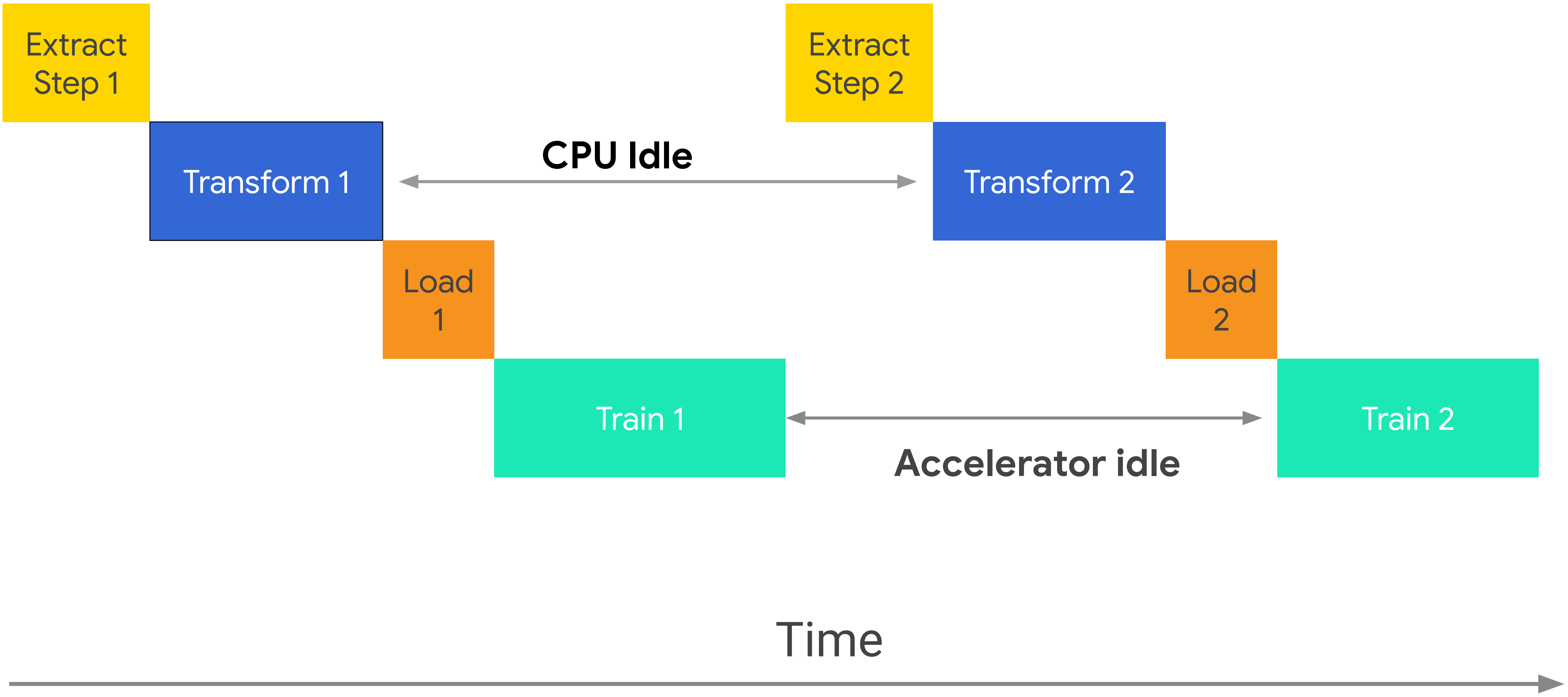
Parallelize across many CPU cores

# # 3. Pipelining with prefetching

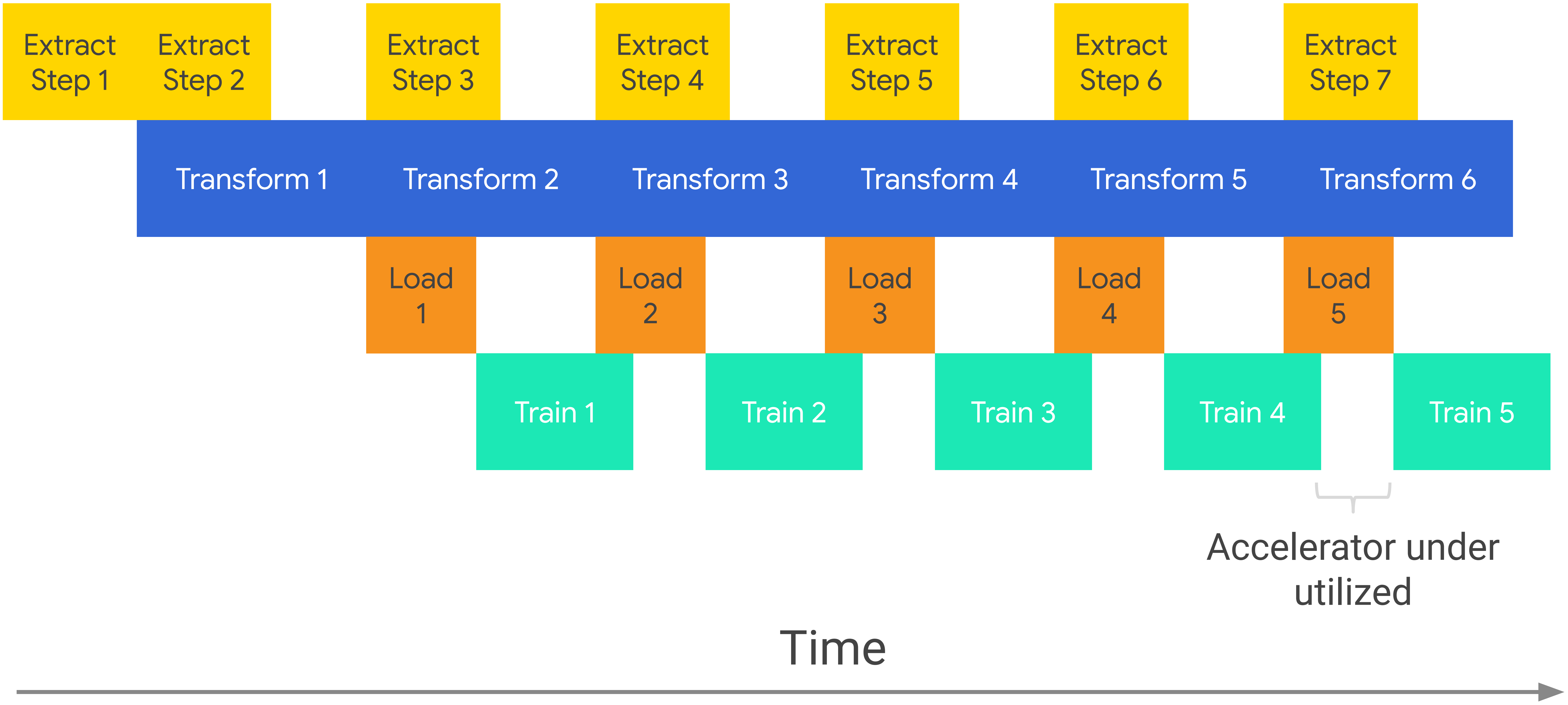
```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
  dataset = dataset.shuffle(buffer_size=10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn, num_parallel_calls=40)
  dataset = dataset.batch(batch_size)
  dataset = dataset.prefetch(buffer_size=1)
  return dataset
```

**Prefetch pipelines everything above with the accelerator training**

# Input pipeline bottleneck



# Updated Input pipeline



# # 4. Using fused transformation ops

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
  dataset = dataset.shuffle(buffer_size=10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn, num_parallel_calls=40)
  dataset = dataset.batch(batch_size)
  dataset = dataset.prefetch(buffer_size=1)
  return dataset
```

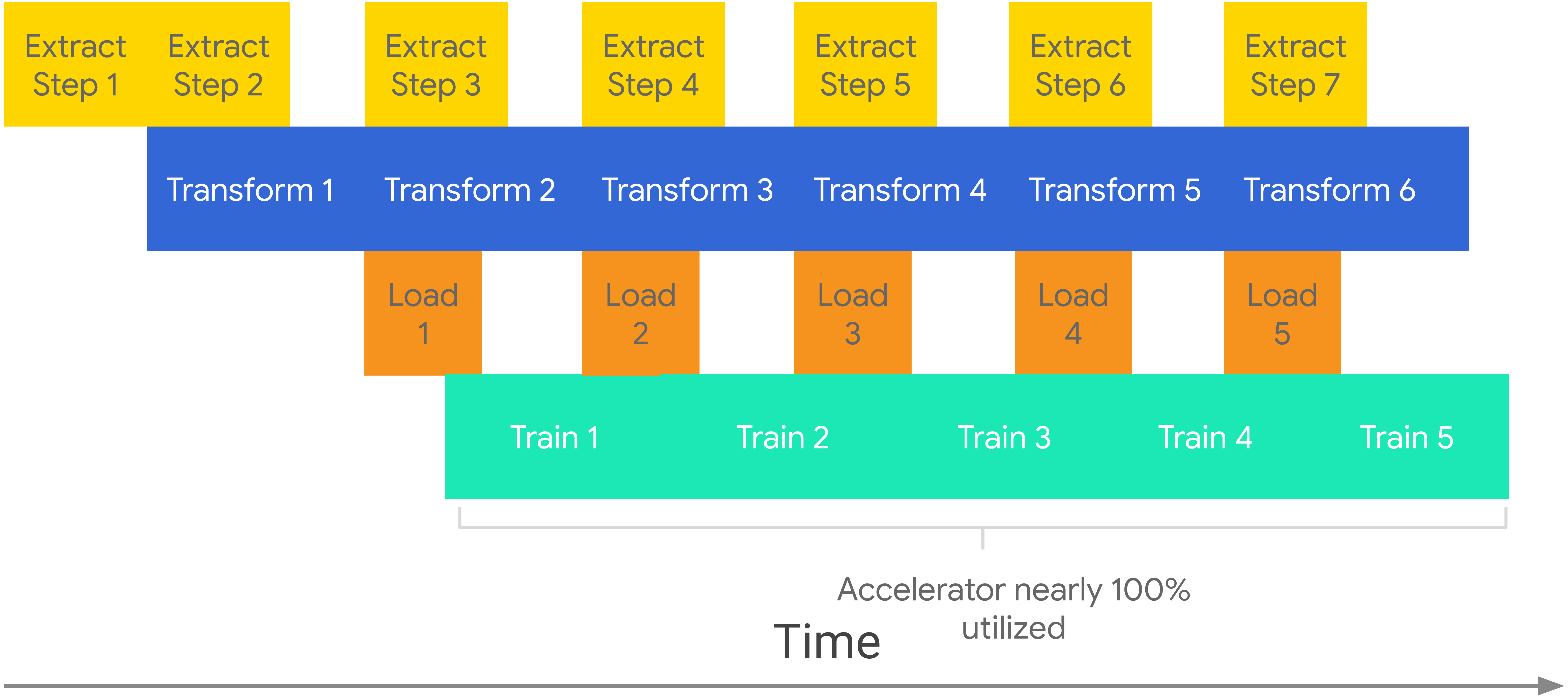
# # 4. Using fused transformation ops

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)
  dataset = dataset.shuffle(buffer_size=10000)
  dataset = dataset.repeat(NUM_EPOCHS)
  dataset = dataset.map(preproc_fn, num_parallel_calls=40)
  dataset = dataset.batch(batch_size)
  dataset = dataset.prefetch(buffer_size=1)
  return dataset
```

# # 4. Using fused transformation ops

```
def input_fn(batch_size):
  files = tf.data.Dataset.list_files(file_pattern)
  dataset = tf.data.TFRecordDataset(files, num_parallel_reads=40)

  dataset = dataset.apply(
      tf.contrib.data.shuffle_and_repeat(buffer_size=10000, NUM_EPOCHS))

  dataset = dataset.apply(
      tf.contrib.data.map_and_batch(parser_fn, batch_size))

  dataset = dataset.prefetch(buffer_size=1)
  return dataset
```

# Updated Input pipeline



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Data parallelism with All Reduce**

Format: On-Camera Screencast

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l10_data_parallelism_with_all_reduce

# Agenda

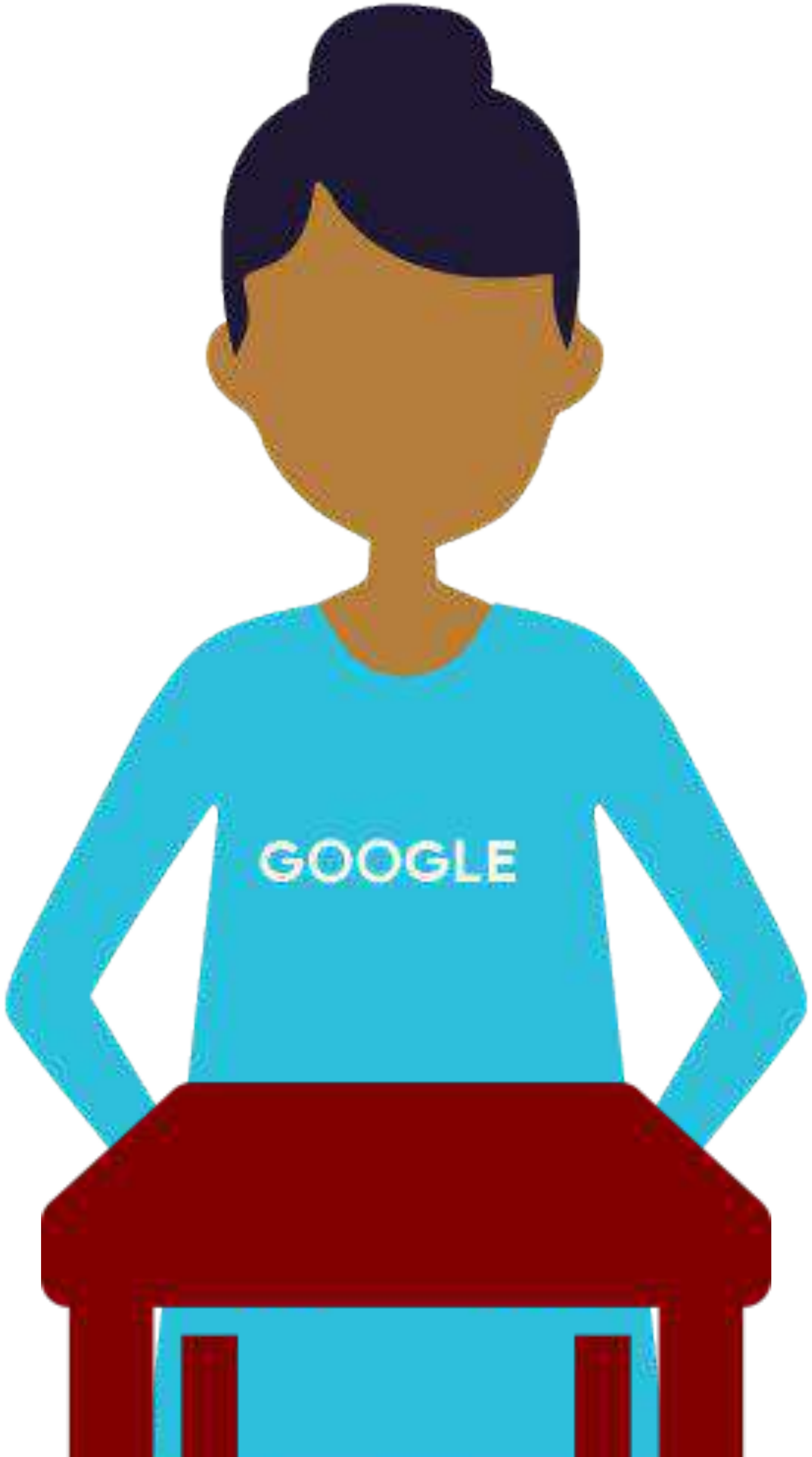
Distributed training

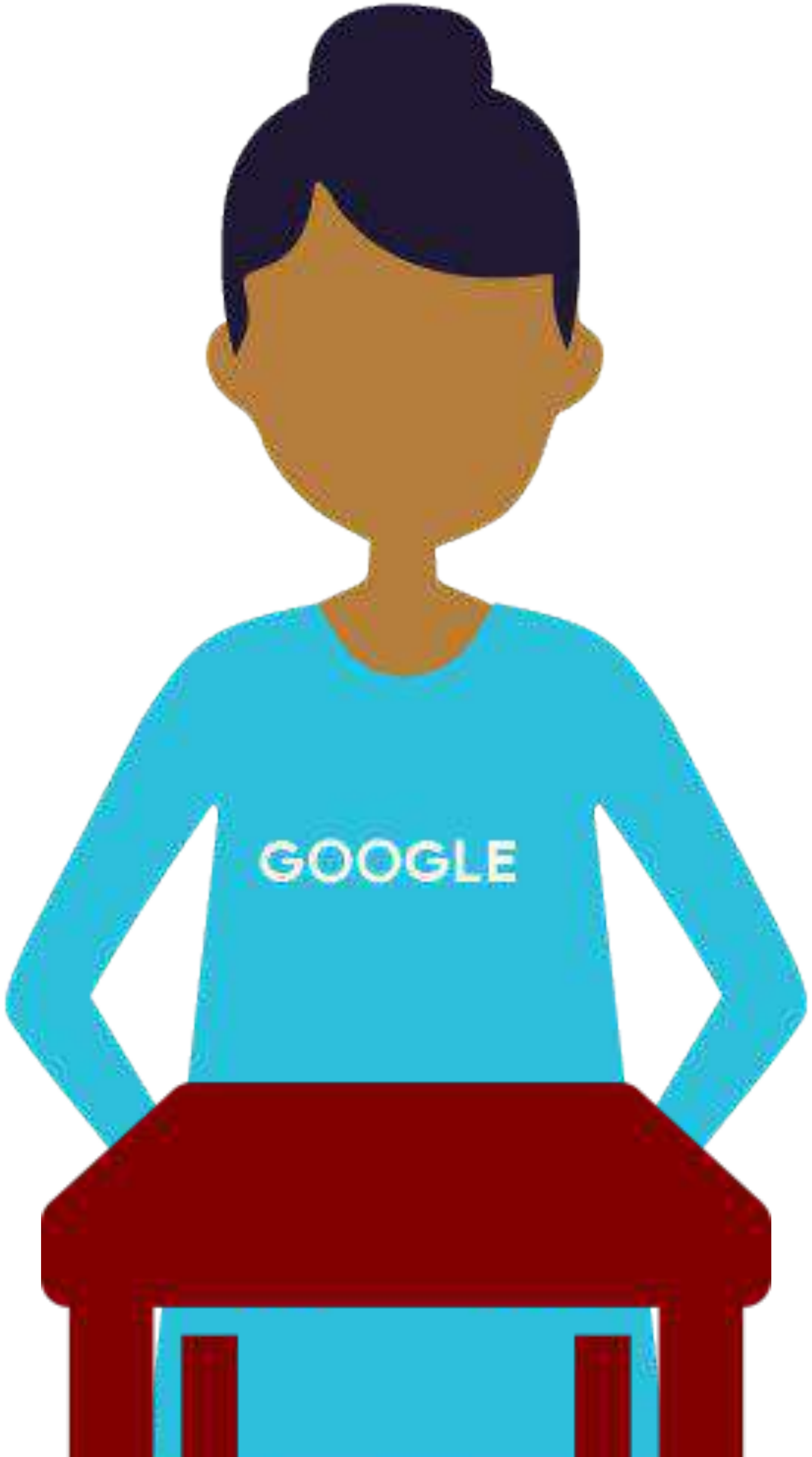
Faster input pipelines

**Data parallelism (All Reduce)**

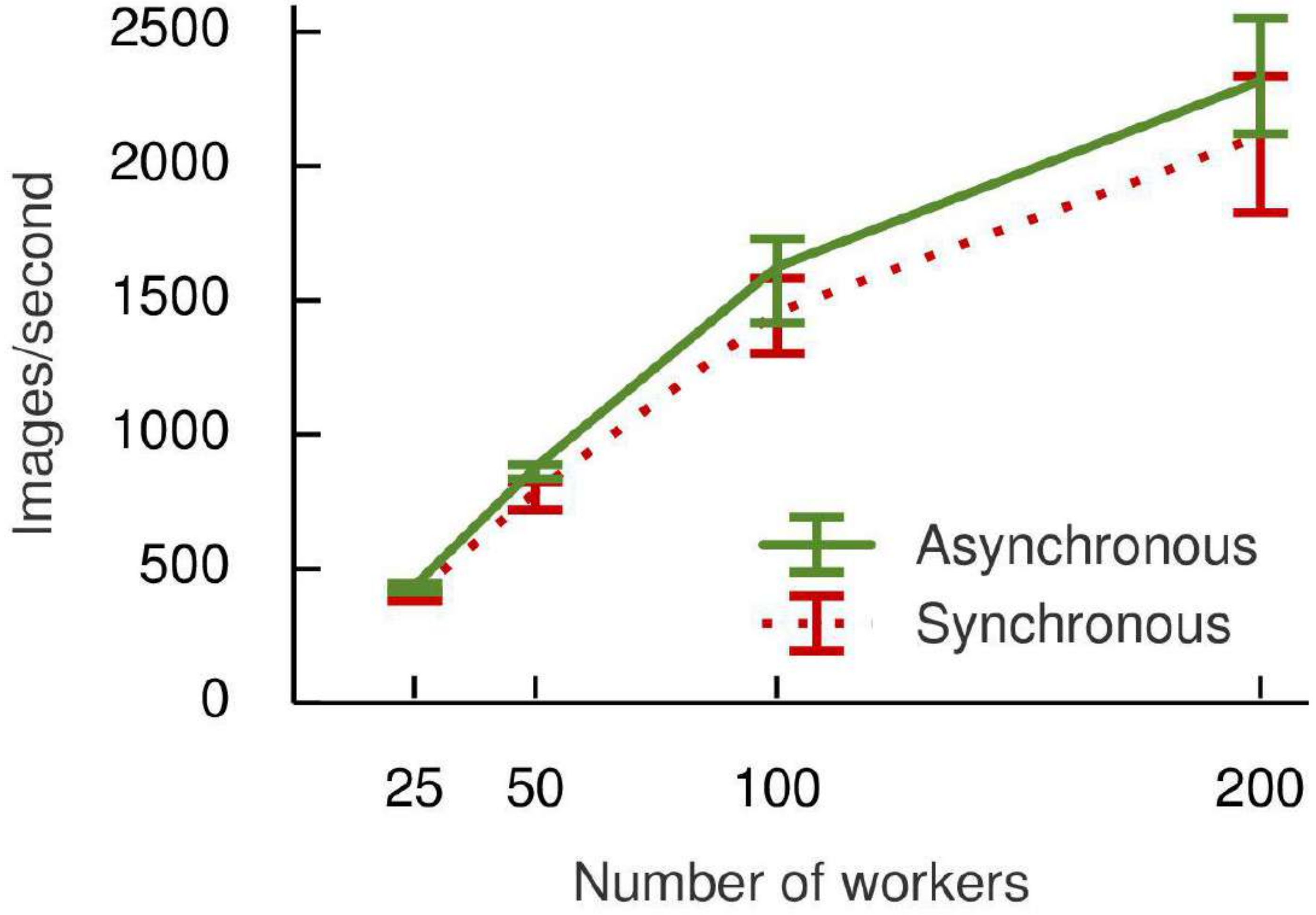
Parameter Server approach

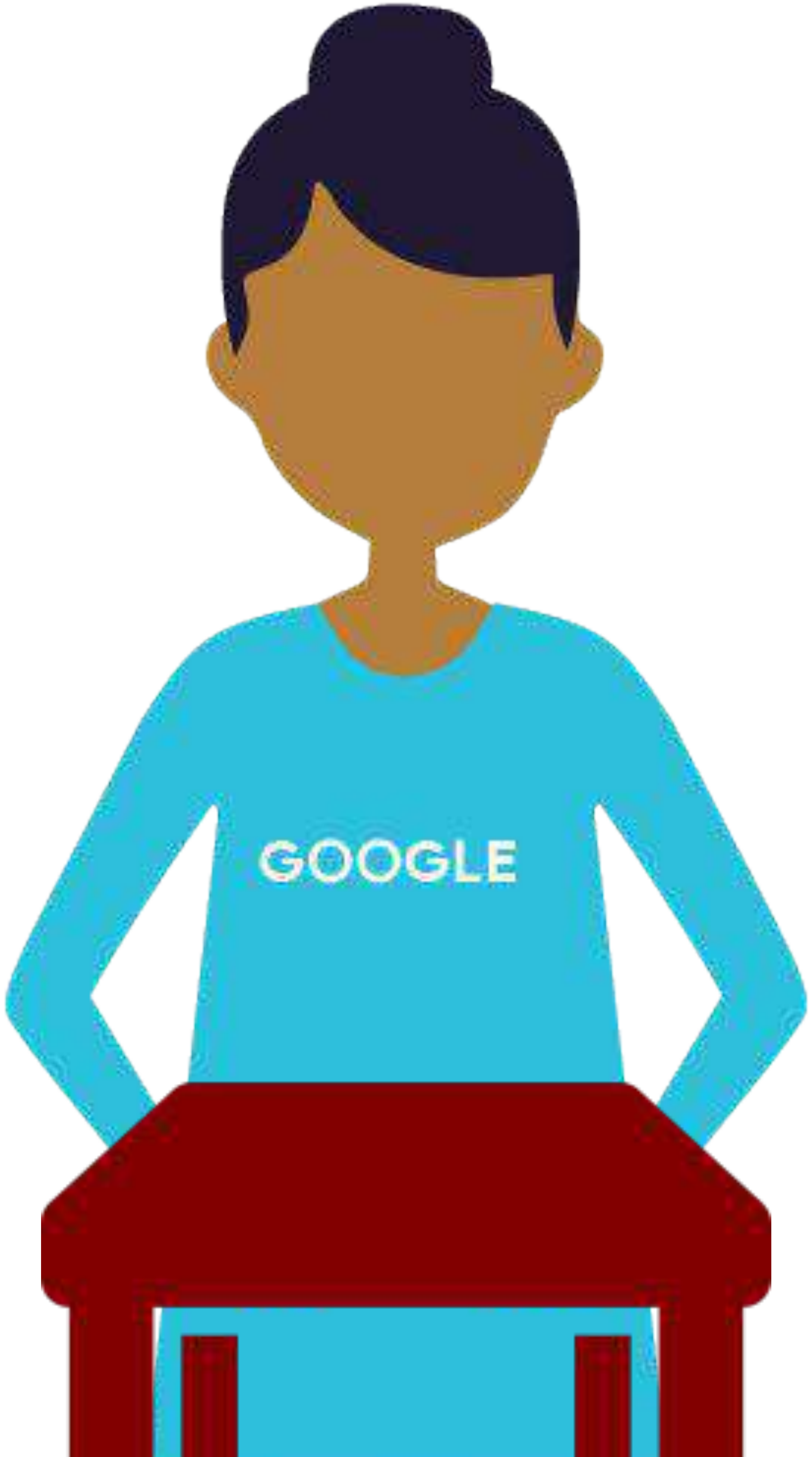
Inference





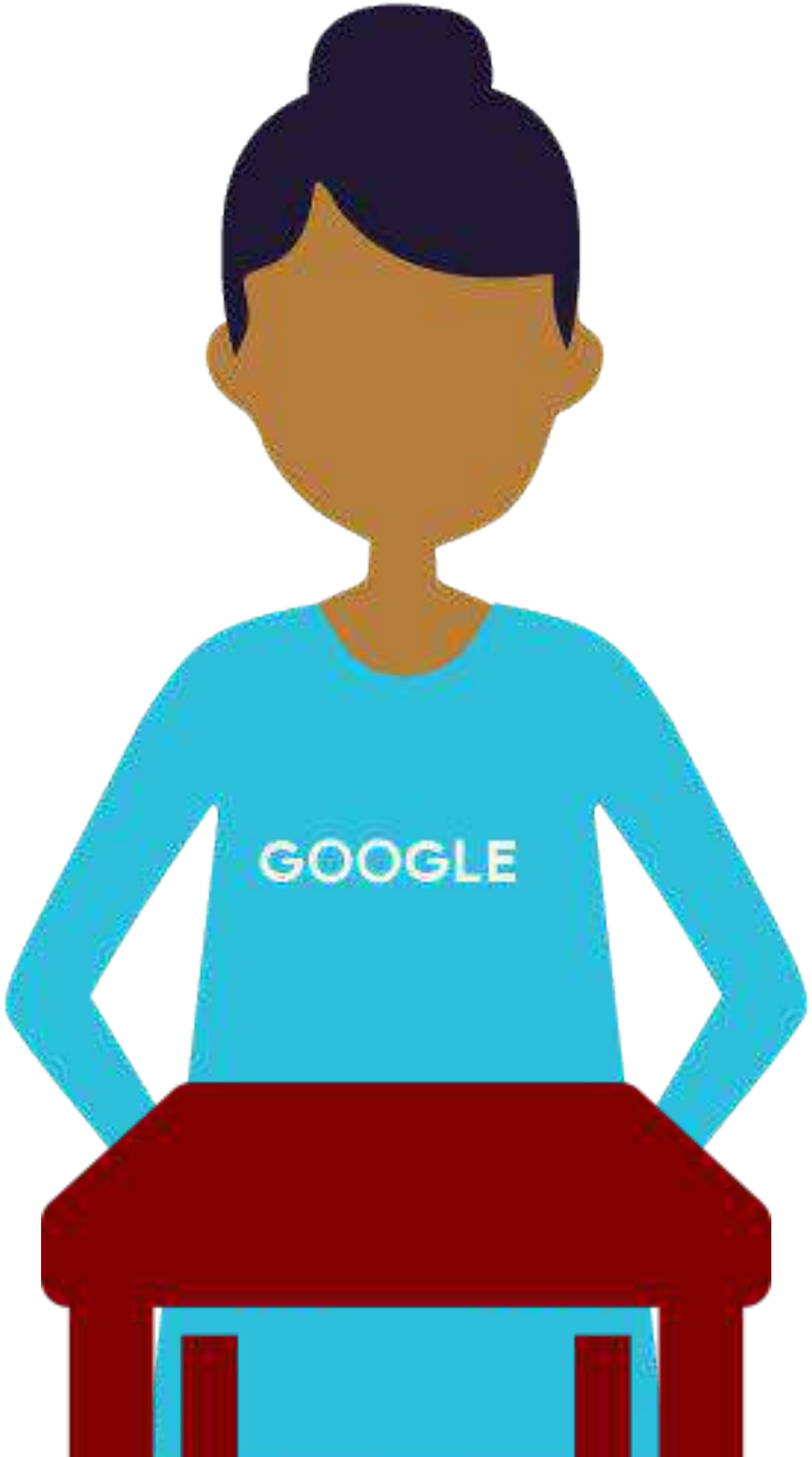
# Data parallelism is a way to increase training throughput






# Distribution API Strategy

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```




# Distribution API Strategy

Easy to use

Fast to train

# Training with Estimator API

```
run_config = tf.estimator.RunConfig()


classifier = tf.estimator.Estimator(
    model_fn=model_function,
    model_dir=model_dir,
    config=run_config)

classifier.train(input_fn=input_function)
```

# Training with Estimator API

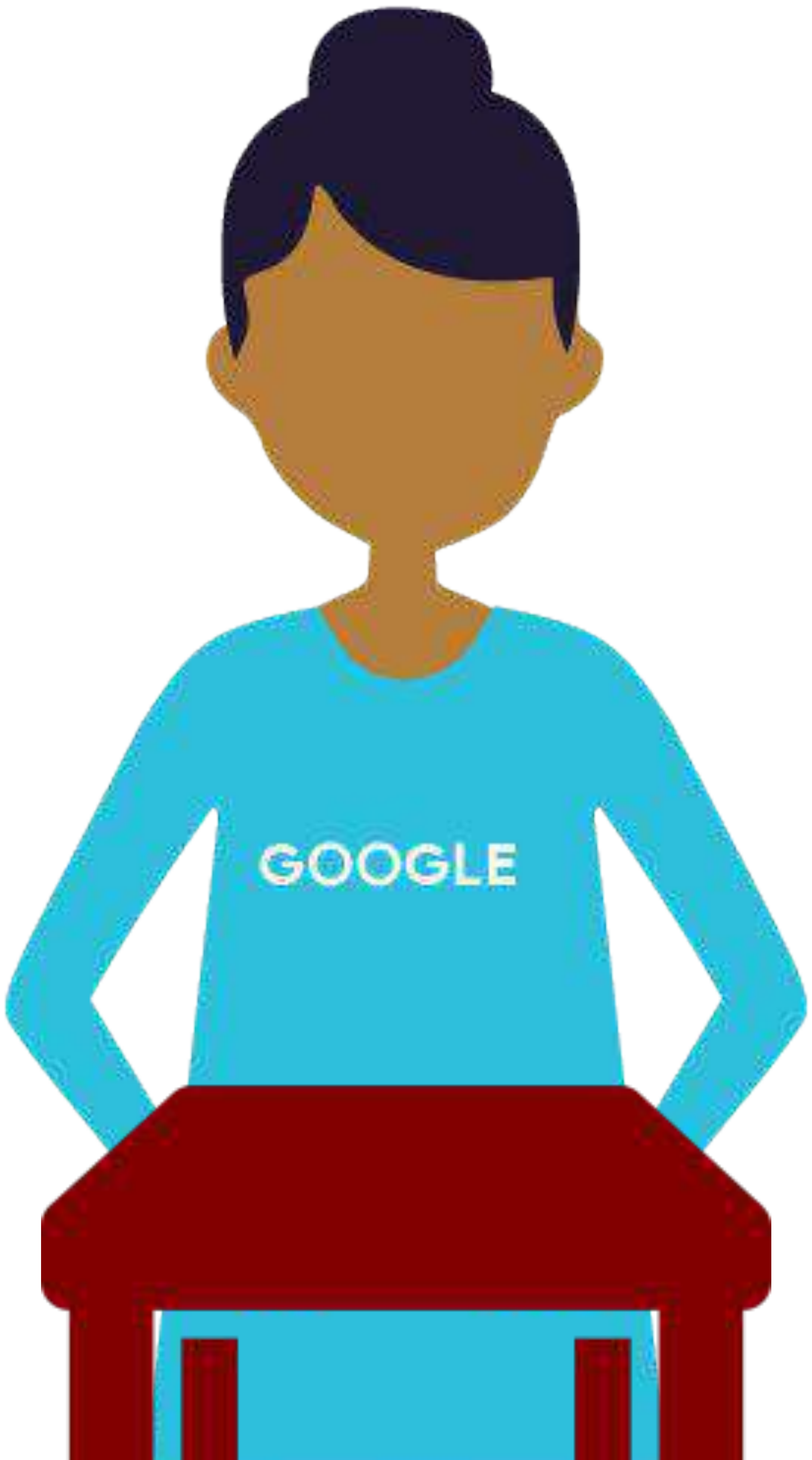
```
distribution = tf.contrib.distribute.MirroredStrategy()
```

**MirroredStrategy for multi GPU distribution**

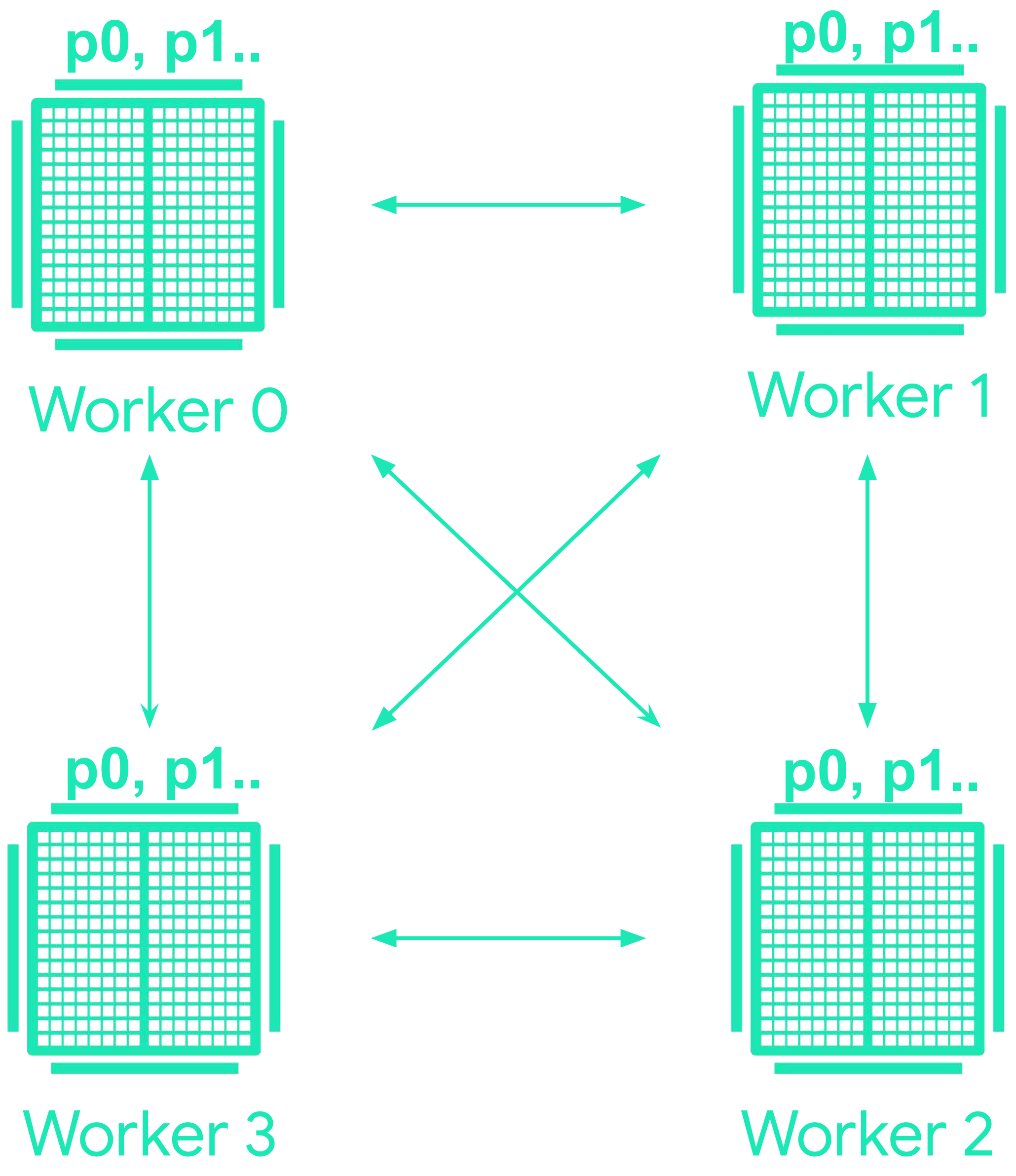
```
run_config = tf.estimator.RunConfig(train_distribute=distribution)
```

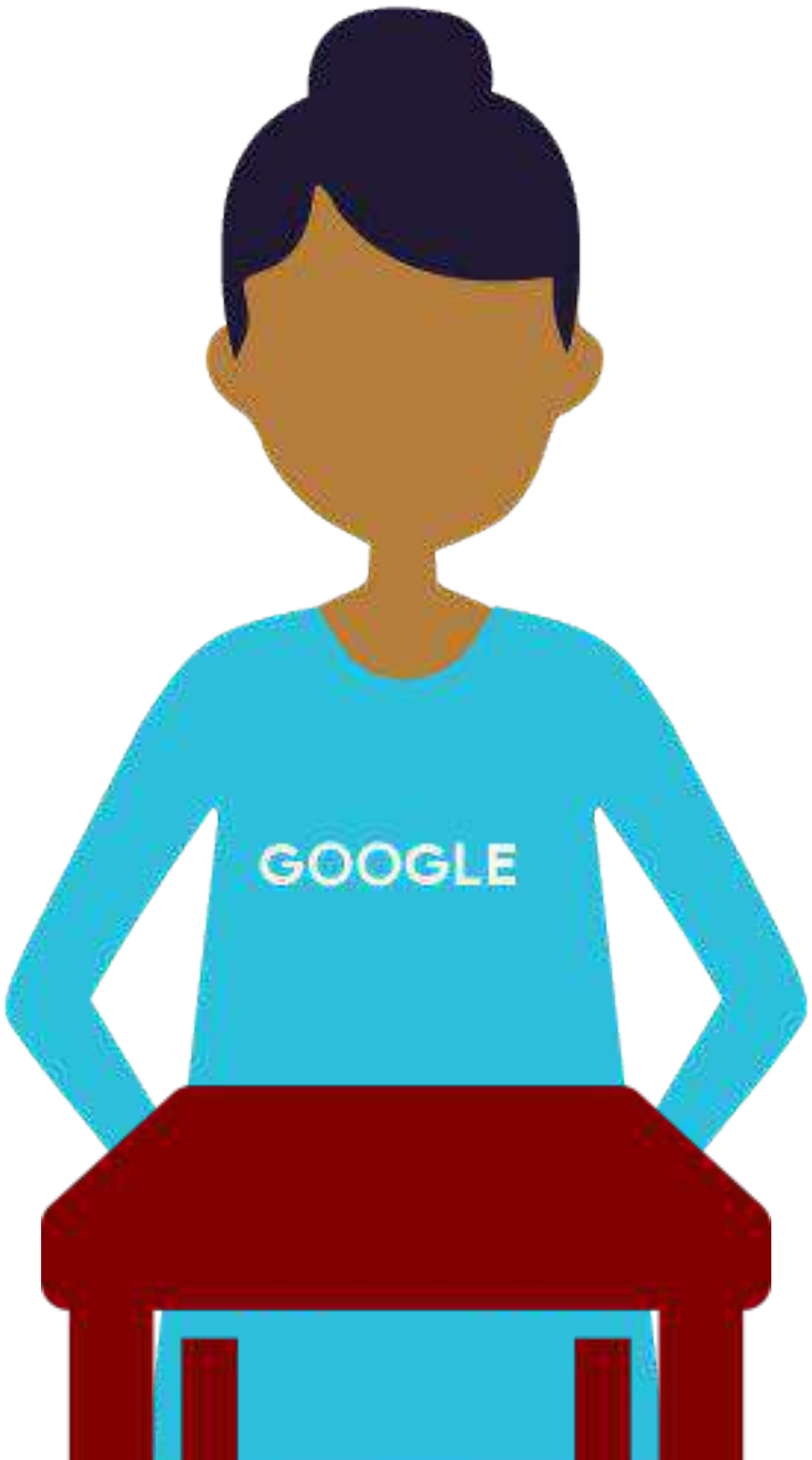
**Pass the distribution to RunConfig**

```
classifier = tf.estimator.Estimator(
    model_fn=model_function,
    model_dir=model_dir,
    config=run_config)

classifier.train(input_fn=input_function)
```




# Mirrored Strategy






# Mirrored Strategy

- No change to the model or training loop
- No change to input function (requires tf.data.Dataset)
- Checkpoints and summaries are seamless

# Mirrored Strategy Demo

Demo goes here

## Training Time

All Submissions

**Objective:** Time taken to train an image classification model to a top-5 validation accuracy of 93% or greater on ImageNet.

| Rank | Time to 93% Accuracy | Model | Hardware | Framework |
|---|---|---|---|---|
| 1 Apr 2018 | 0:30:43 | ResNet50 *Google* source | Half of a TPUv2 Pod | TensorFlow 1.8.0-rc1 |
| 2 Apr 2018 | 1:06:32 | AmoebaNet-D N6F256 *Google* source | 1/4 of a TPUv2 Pod | TensorFlow 1.8.0-rc1 |
| 3 Apr 2018 | 1:58:24 | AmoebaNet-D N6F256 *Google* source | 1/16 of a TPUv2 Pod | TensorFlow 1.8.0-rc1 |
| 4 Apr 2018 | 2:57:28 | Resnet 50 *fast.ai + students team: Jeremy Howard, Andrew Shaw, Brett Koonce, Sylvain Gugger* source | 8 * V100 (AWS p3.16xlarge) | fastai / pytorch |
| 5 Apr 2018 | 3:25:55 | ResNet50 *Intel(R) Corporation* source | 128 nodes with Xeon Platinum 8124M / 144 GB / 36 Cores (Amazon EC2 [c5.18xlarge]) | Intel(R) Optimized Caffe |

https://dawn.cs.stanford.edu/benchmark/

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Parameter Server Approach**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l11_parameter_server_approach

# Agenda

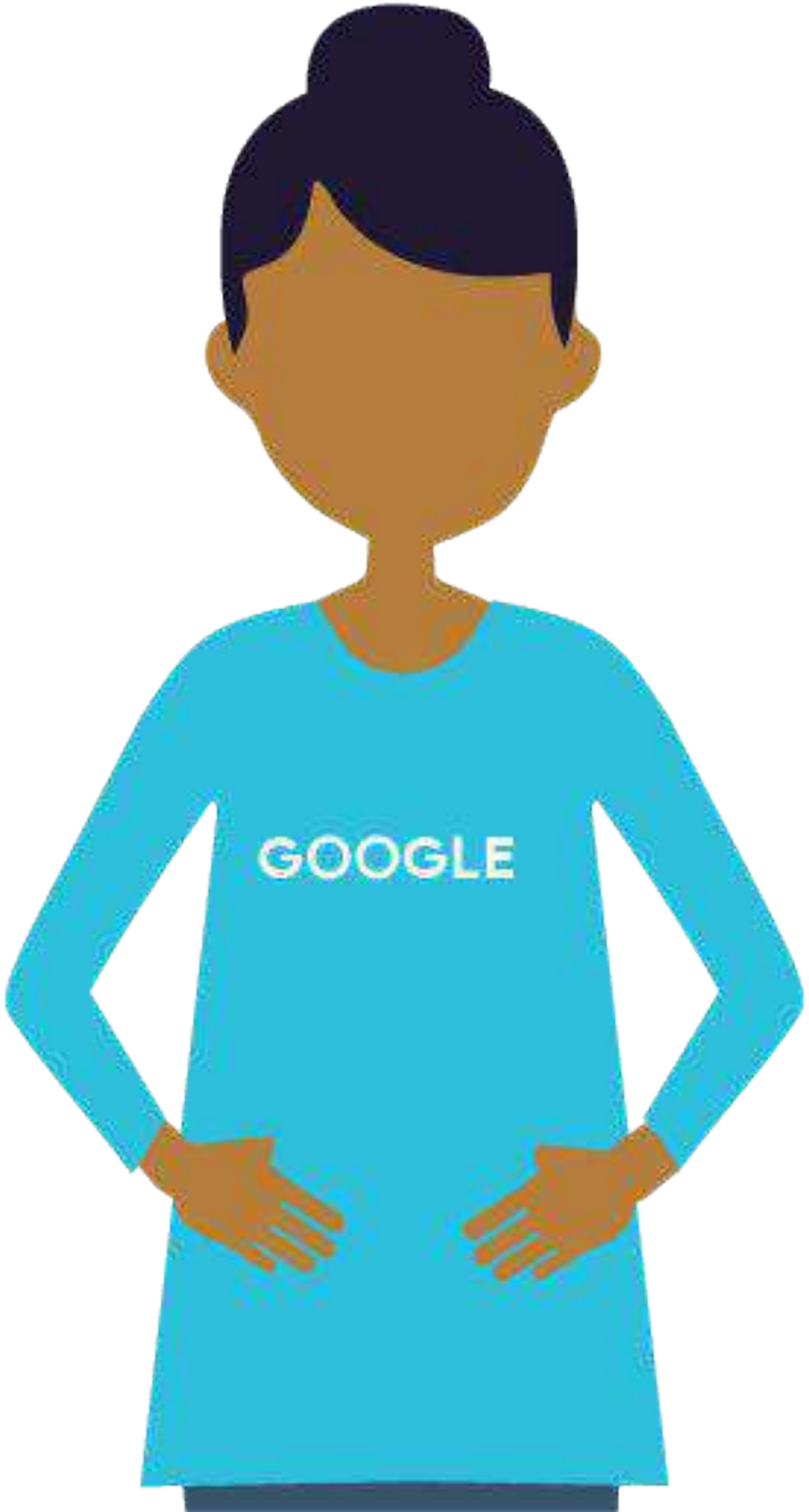
Distributed training

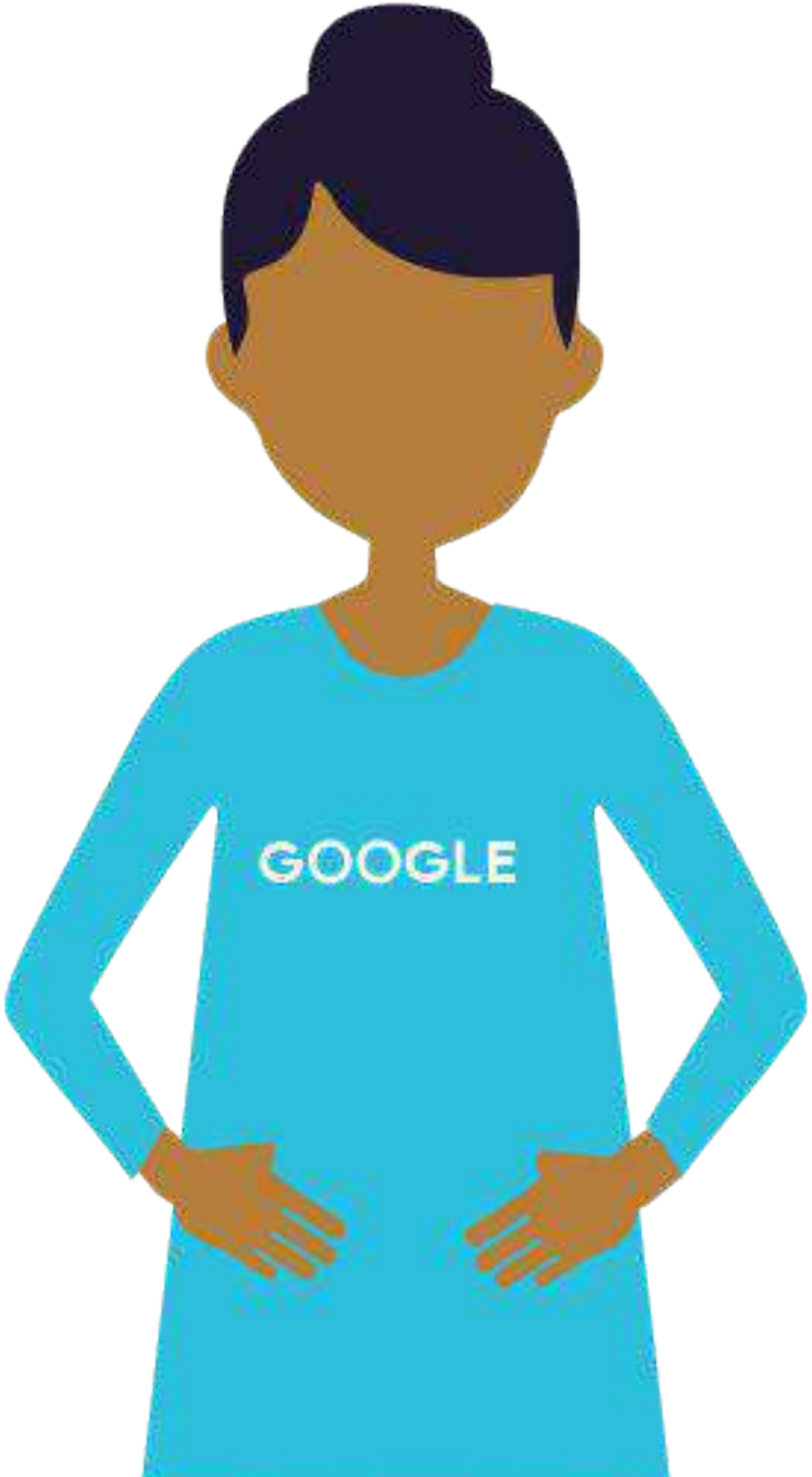
Faster input pipelines

Data parallelism (All Reduce)

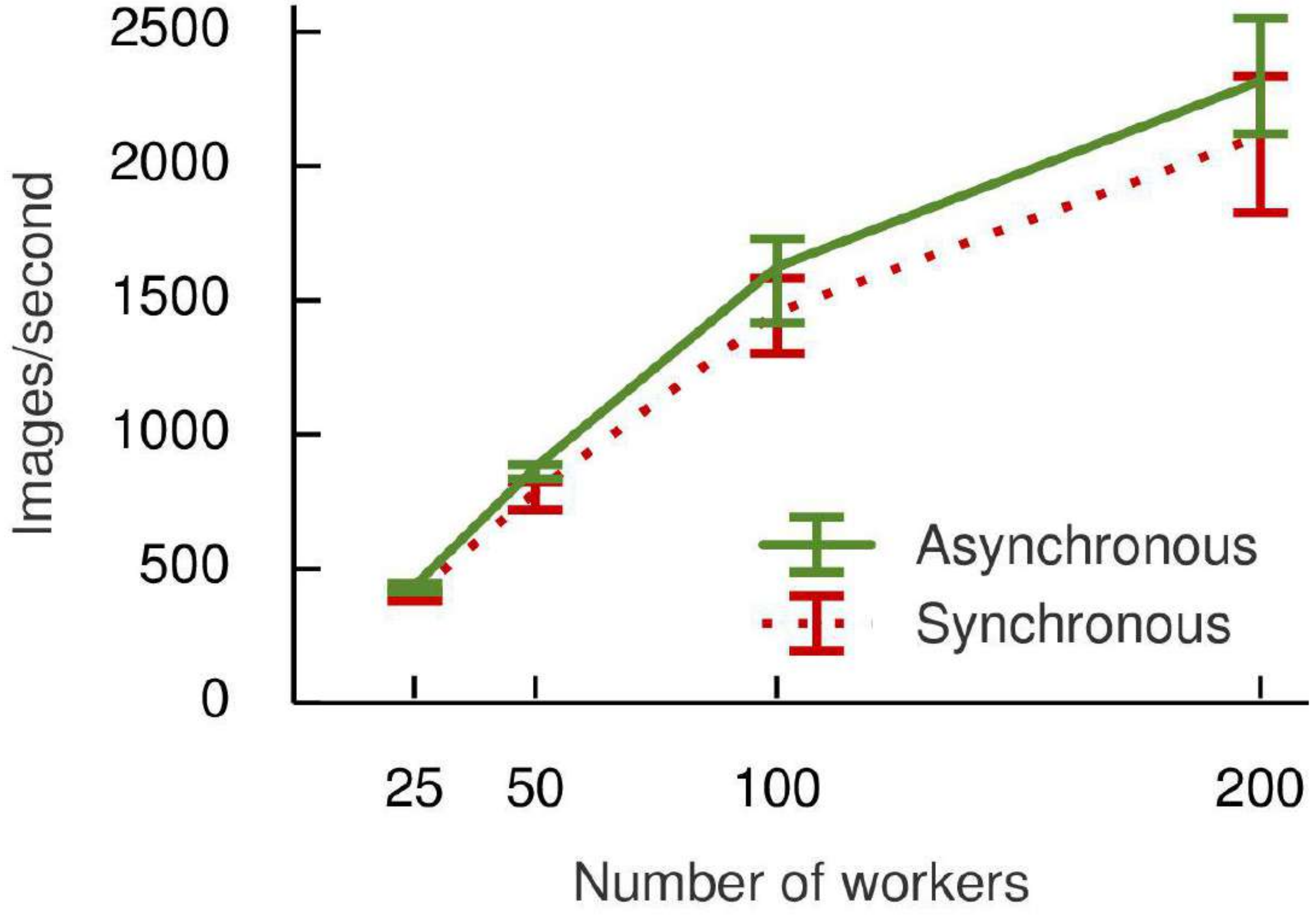
**Parameter Server approach**

Inference

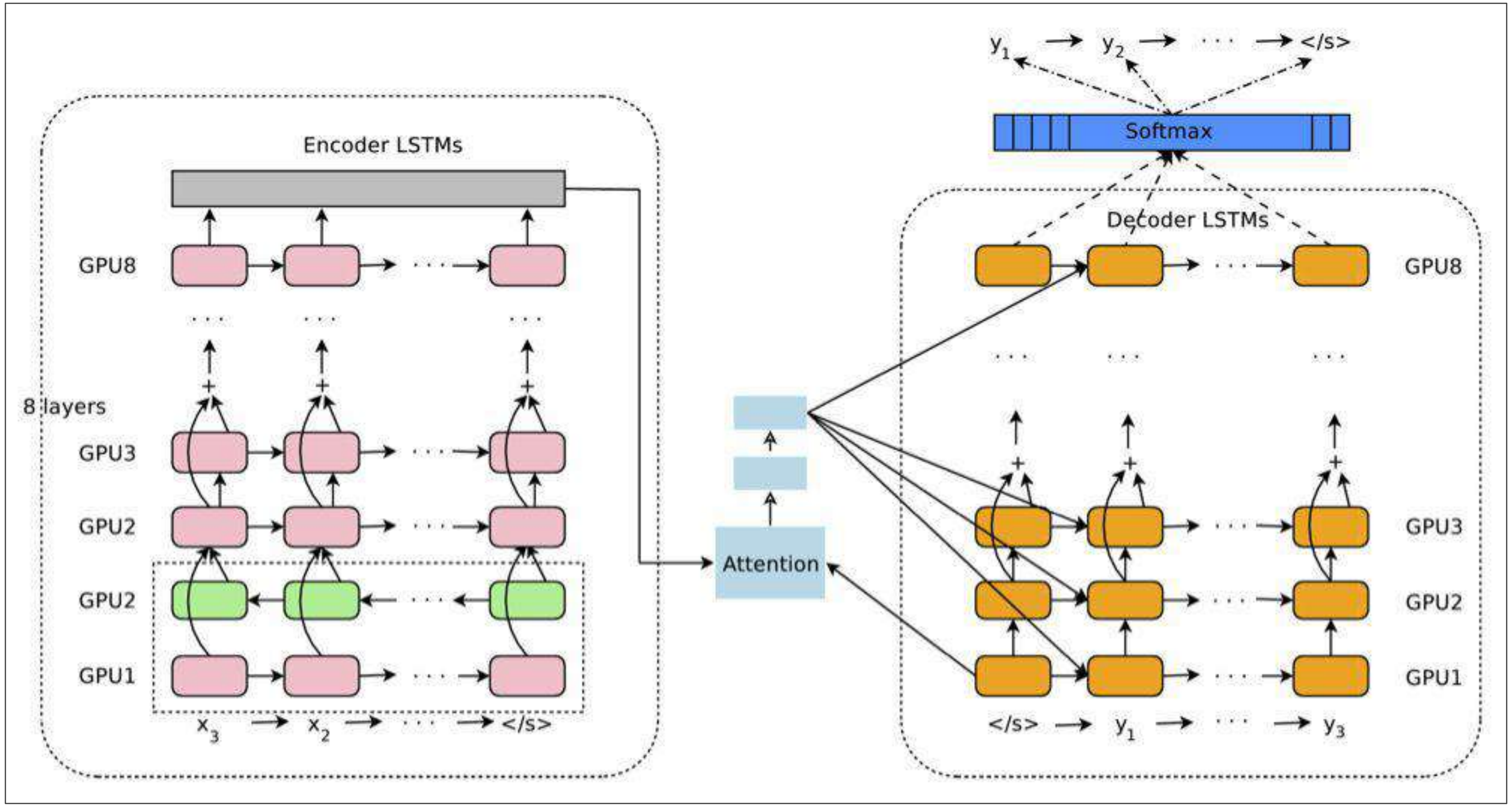


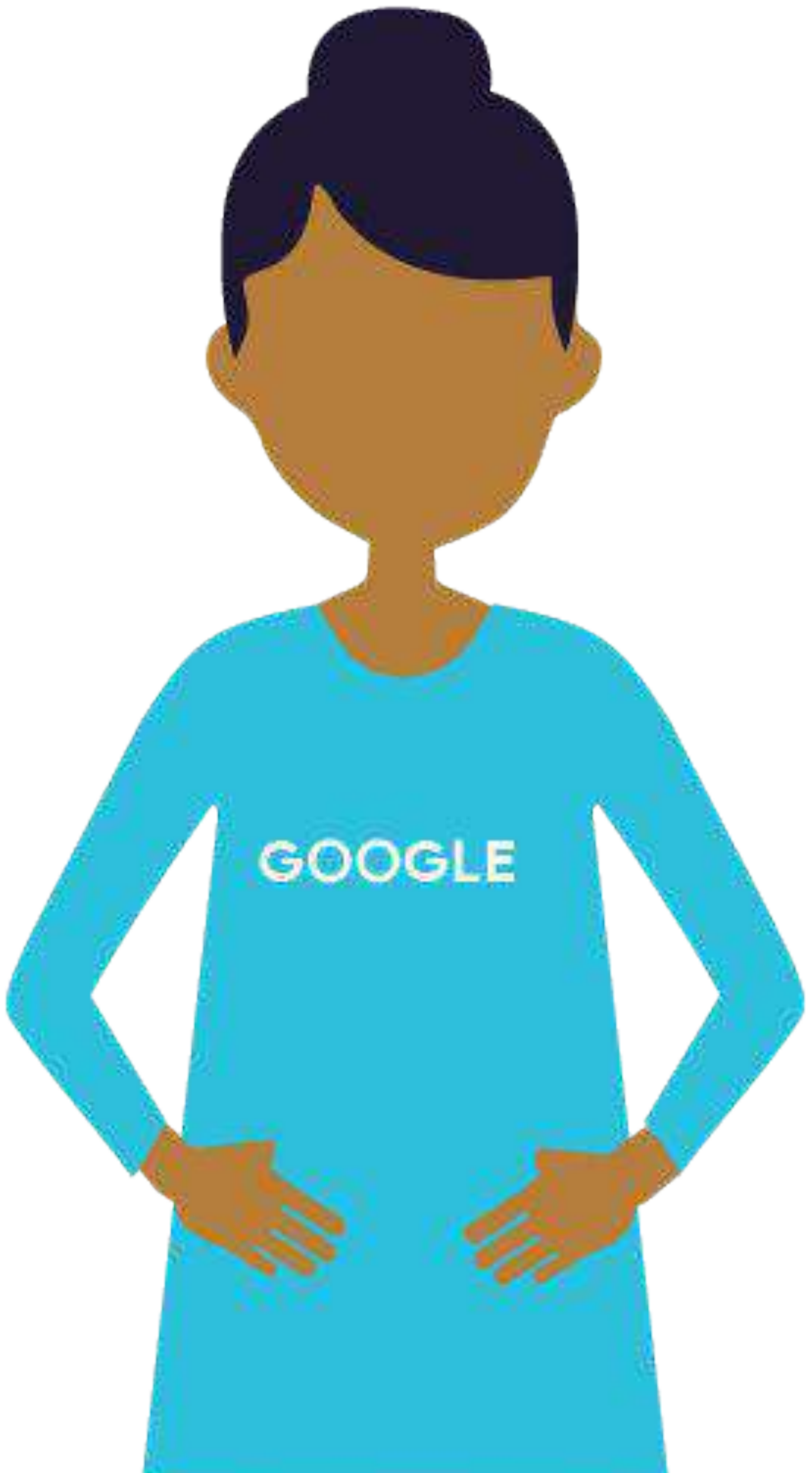


# Data parallelism is a way to increase training throughput

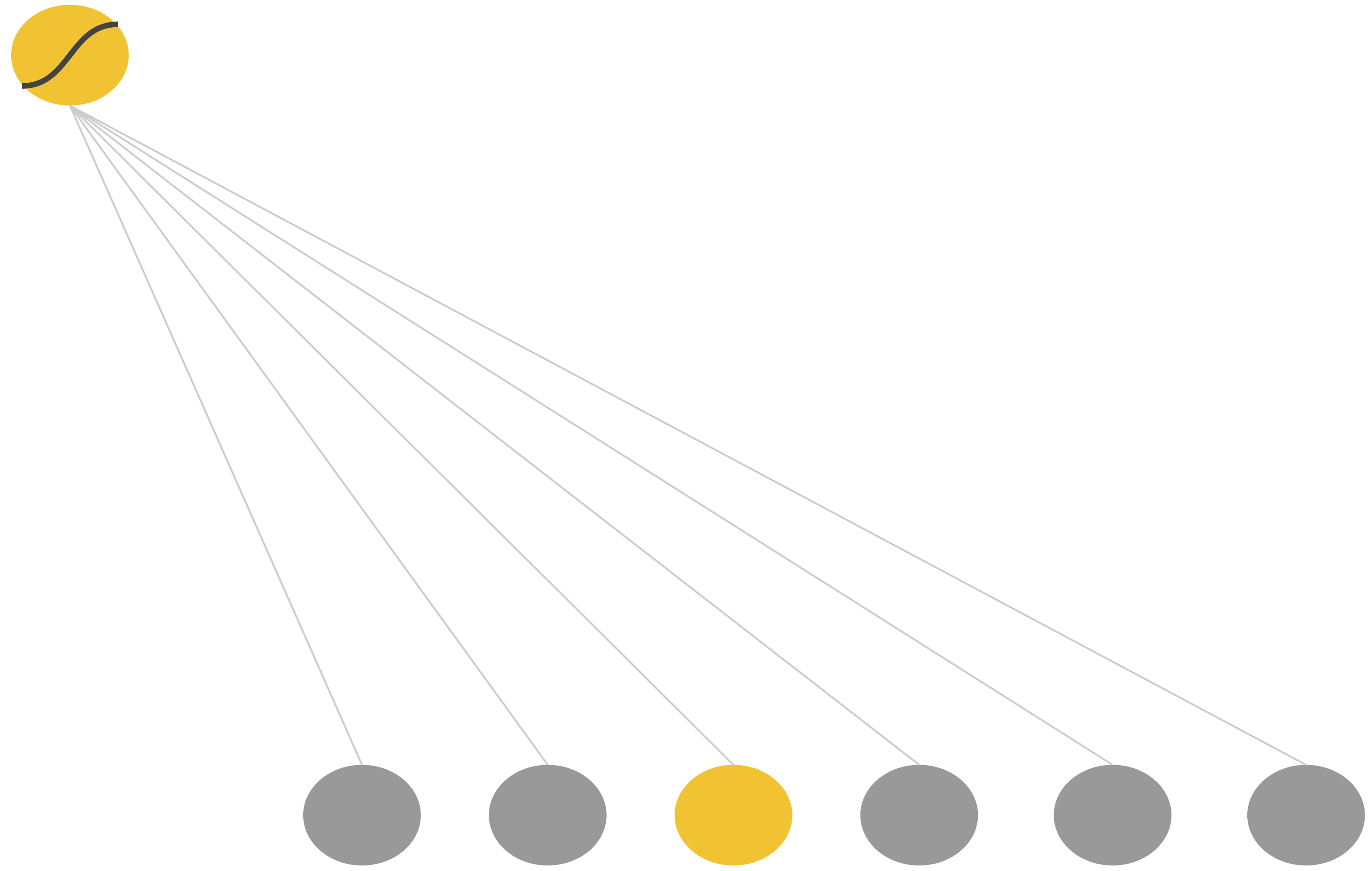


# Model parallelism lets you distribute a model across GPUs

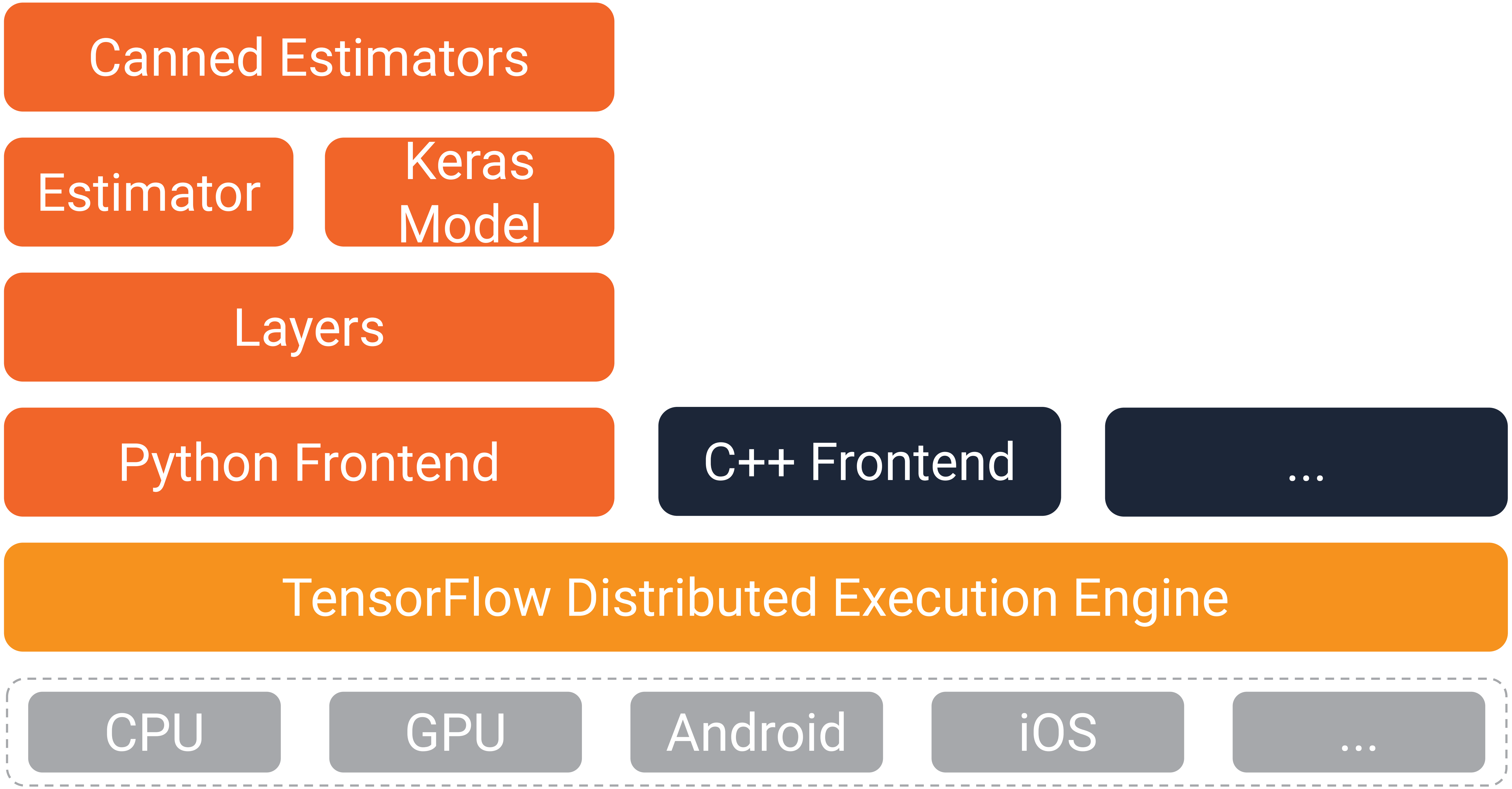




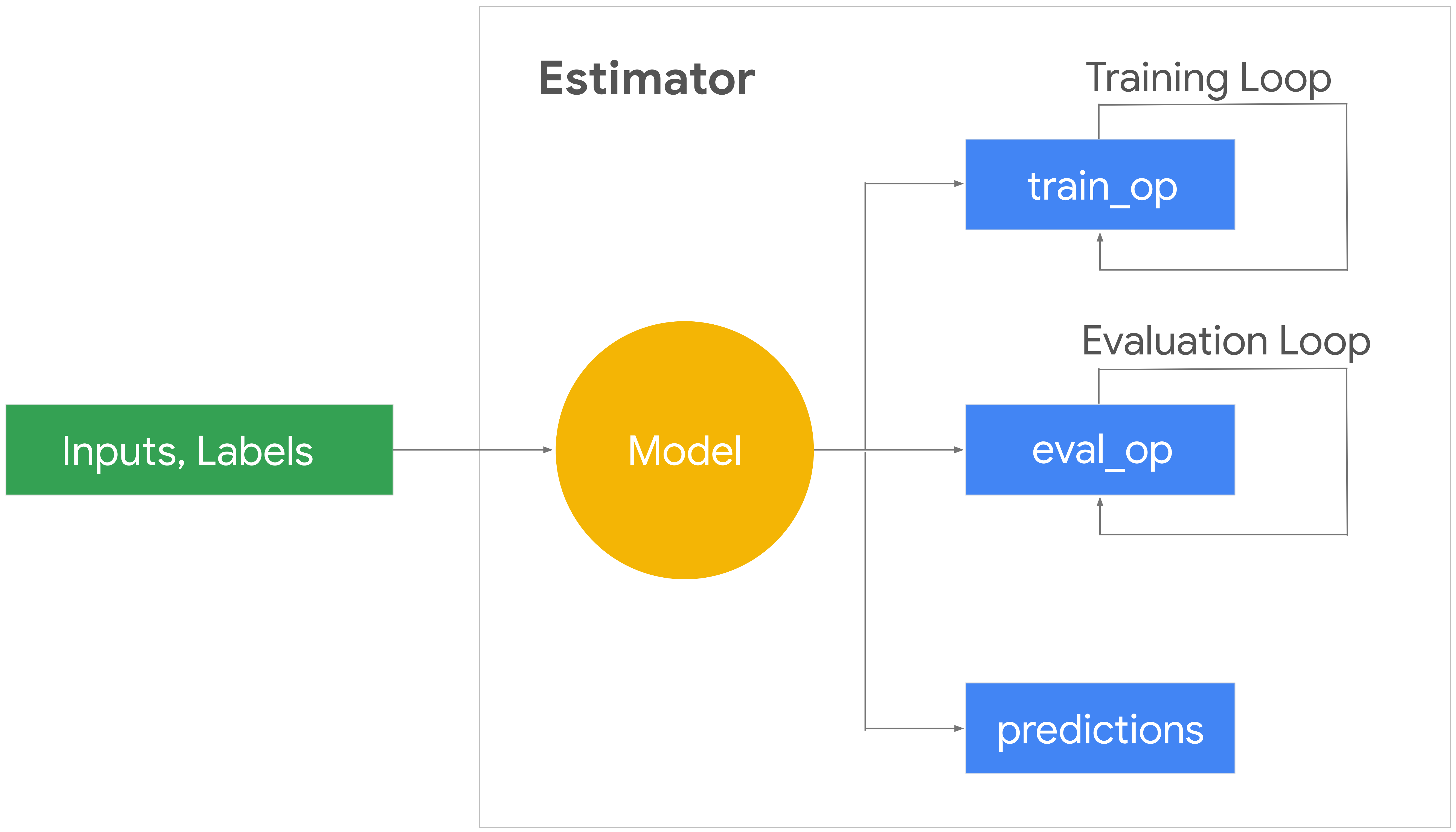

Large embeddings need multiple machines to map sparse data



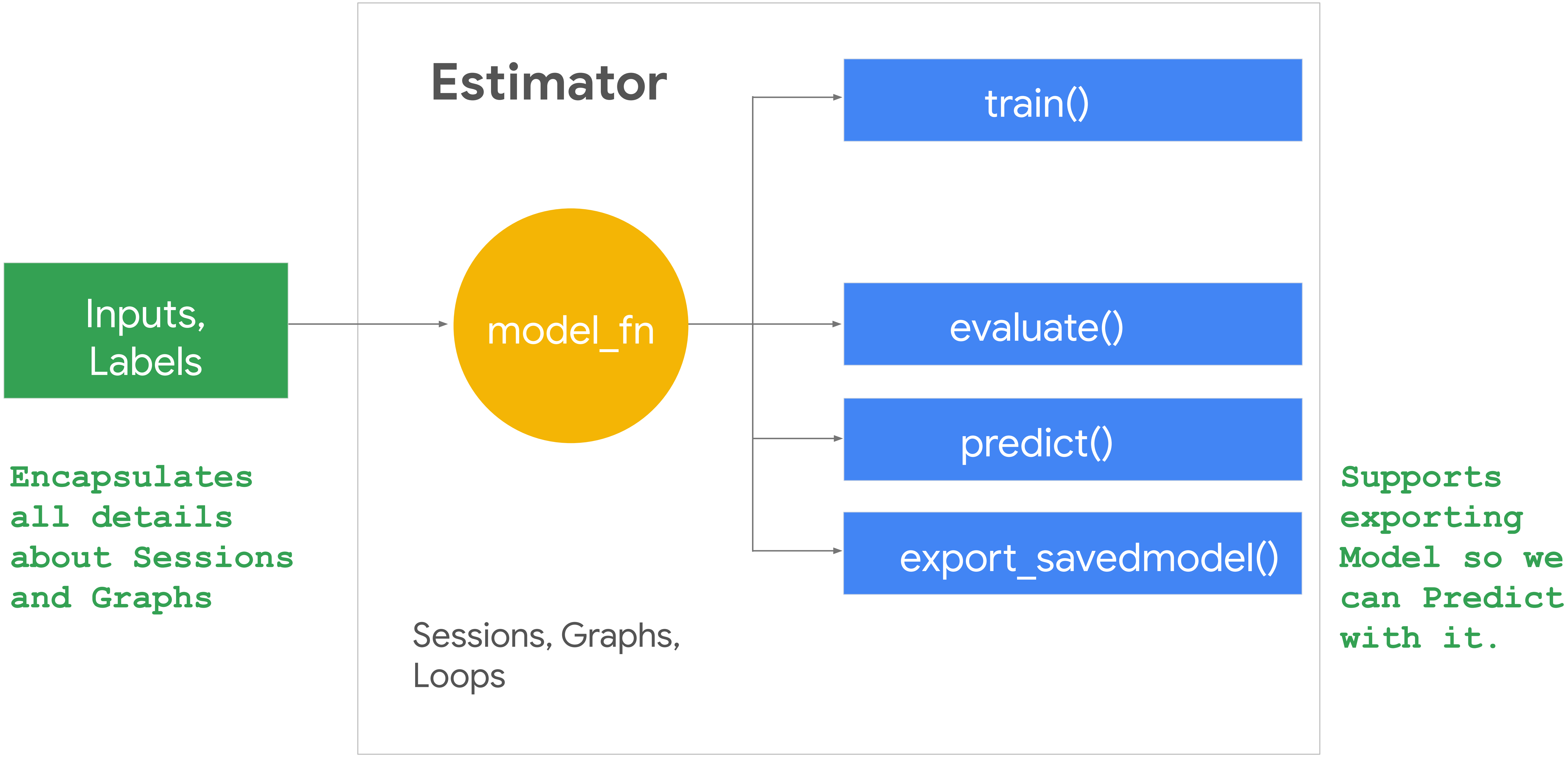
# Estimator train_and_evaluate() handles all this



# Estimator contains the implementation of three functions



By encapsulating details about sessions and graphs, it also supports exporting the model for serving



# train_and_evaluate bundles together a distributed workflow

```
def train_and_evaluate(output_dir, config, params):
  features = [tf.feature_column.embedding_column(...),
              tf.feature_column.bucketized_column(...)]
  estimator = tf.estimator.Estimator(model_fn = simple_rnn,
                                     model_dir = output_dir)
  train_spec = tf.estimator.TrainSpec(input_fn = get_train(),
                                      max_steps = 1000)
  exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
  eval_spec = tf.estimator.EvalSpec(input_fn = get_valid(),
                                    steps = None,
                                    exporters = exporter)
  tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)


train_and_evaluate(output_dir)
```

Runs training, evaluation, etc. on Cloud ML

Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Inference**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l12_inference

# Agenda

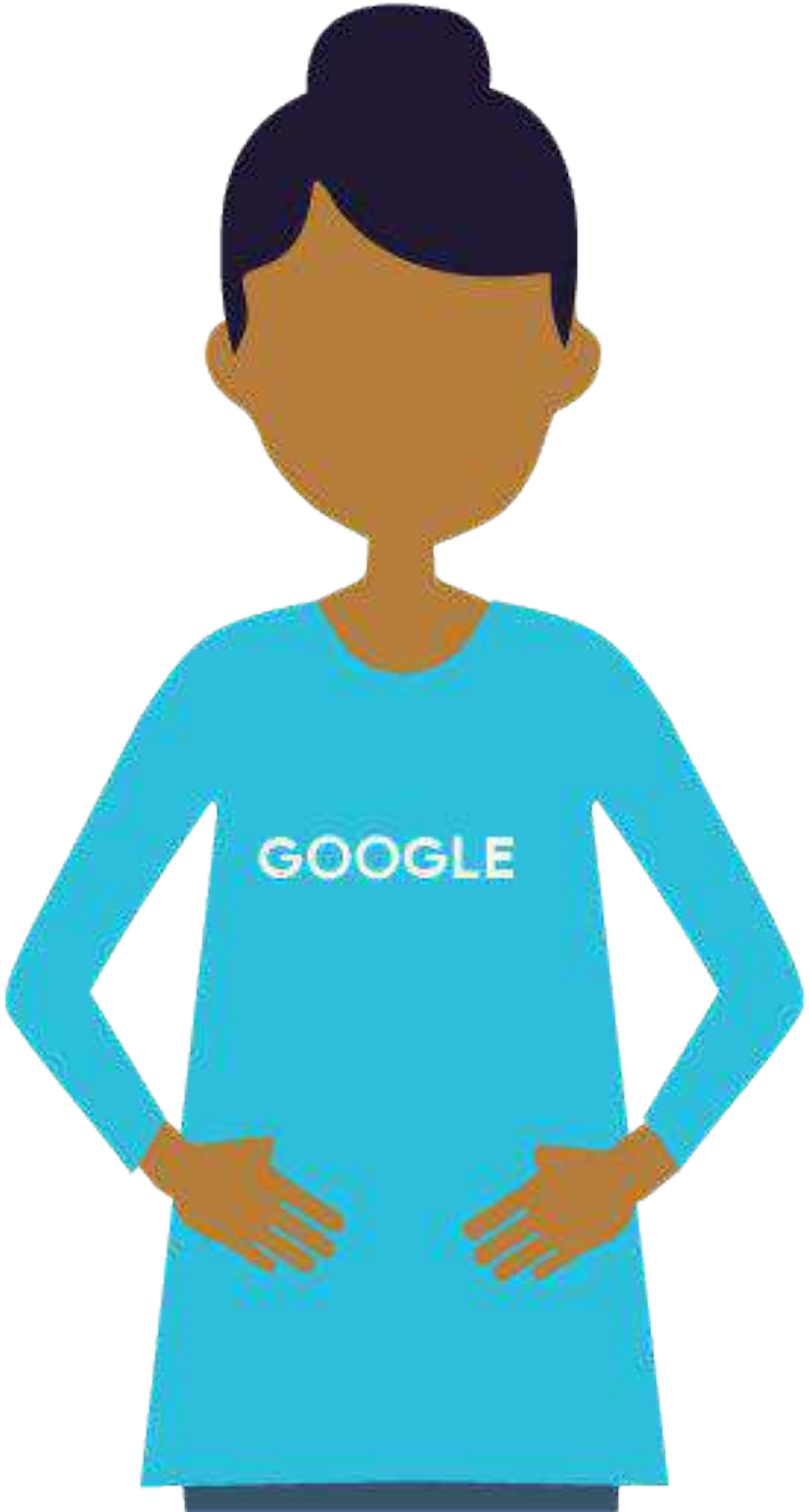
Distributed training

Faster input pipelines

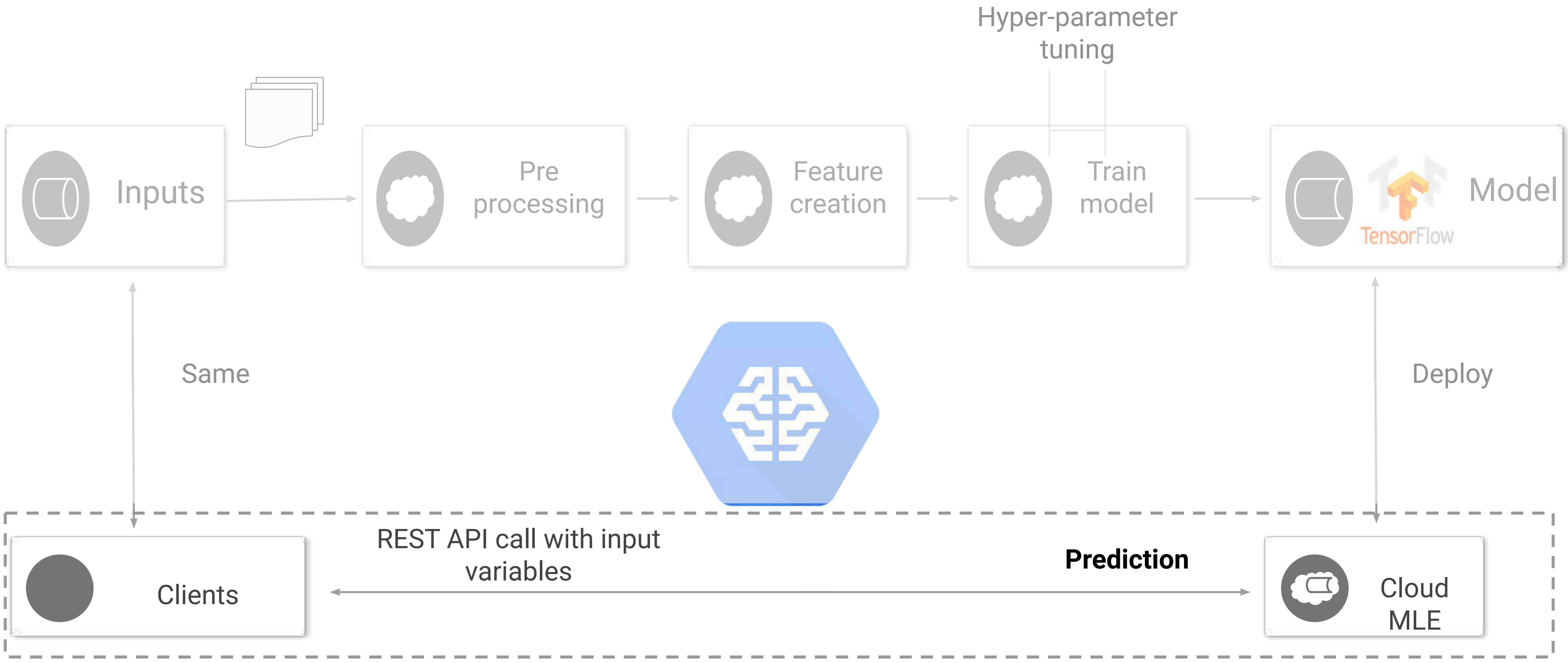
Data parallelism (All Reduce)

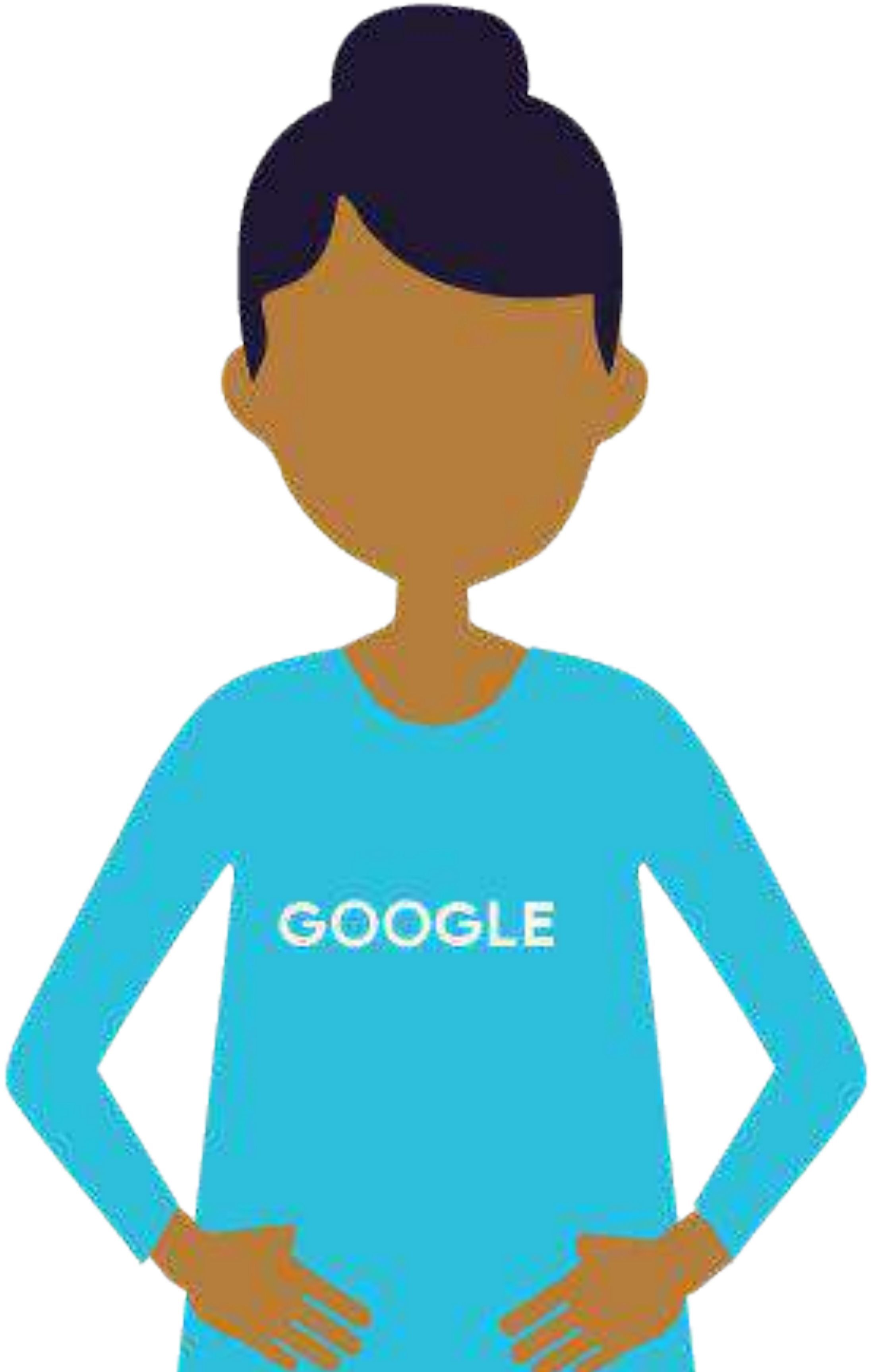
Parameter Server approach

**Inference**

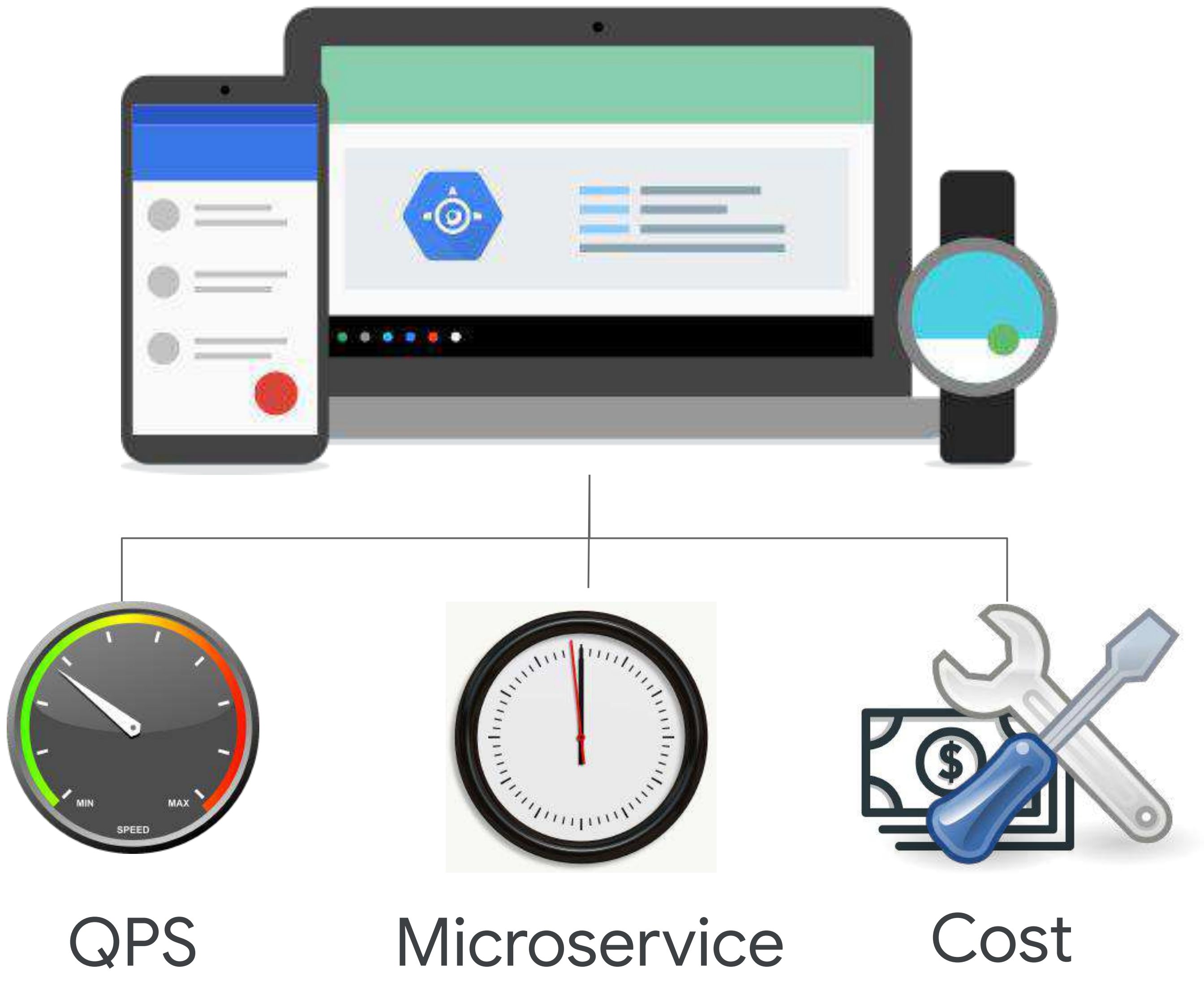


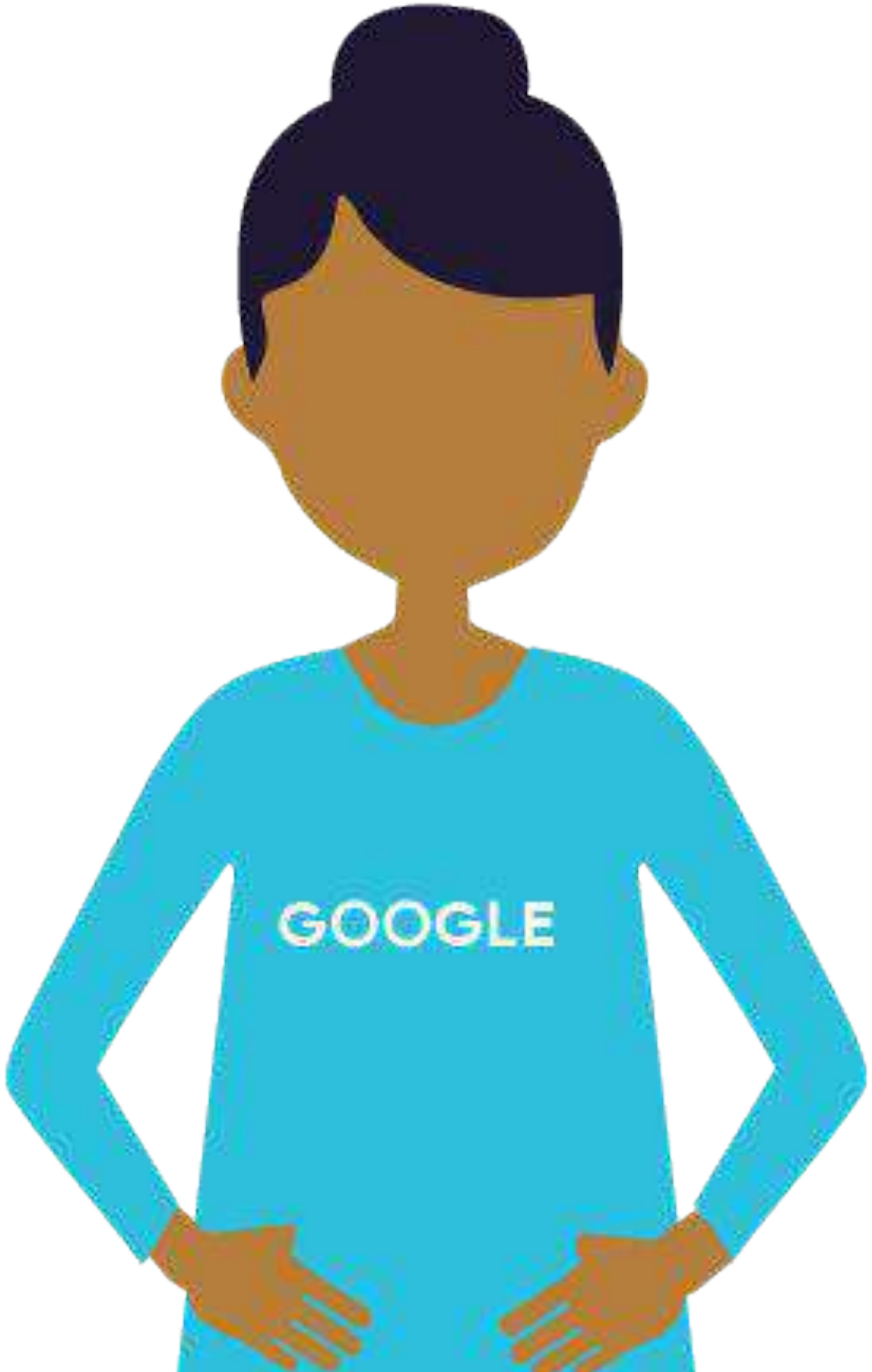
# Performance must consider prediction-time, not just training






# Aspects of performance during inference

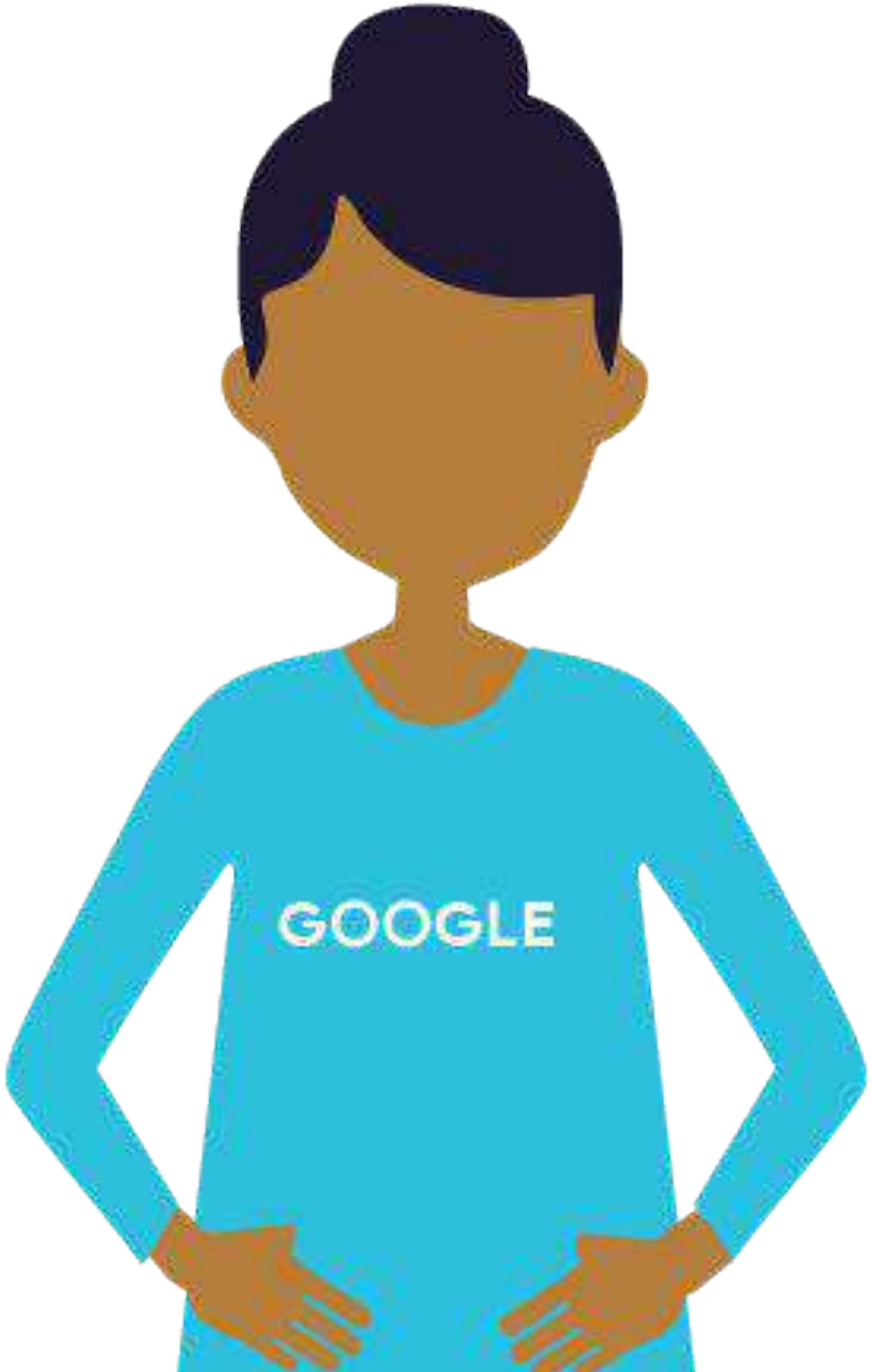





# Implementation Options

REST/HTTP
API

For Streaming
Pipelines



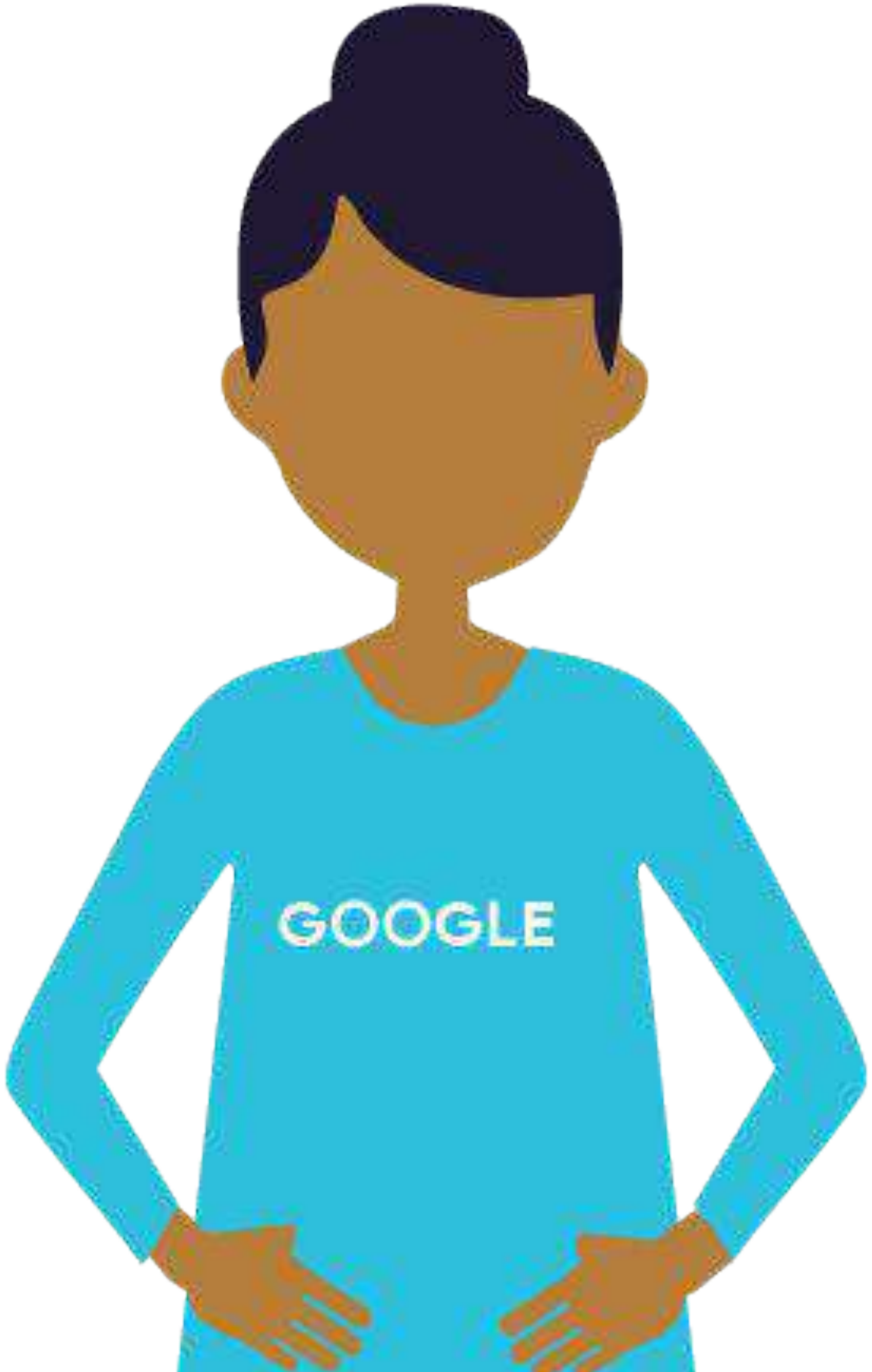

# Implementation Options

REST/HTTP API

For Streaming Pipelines

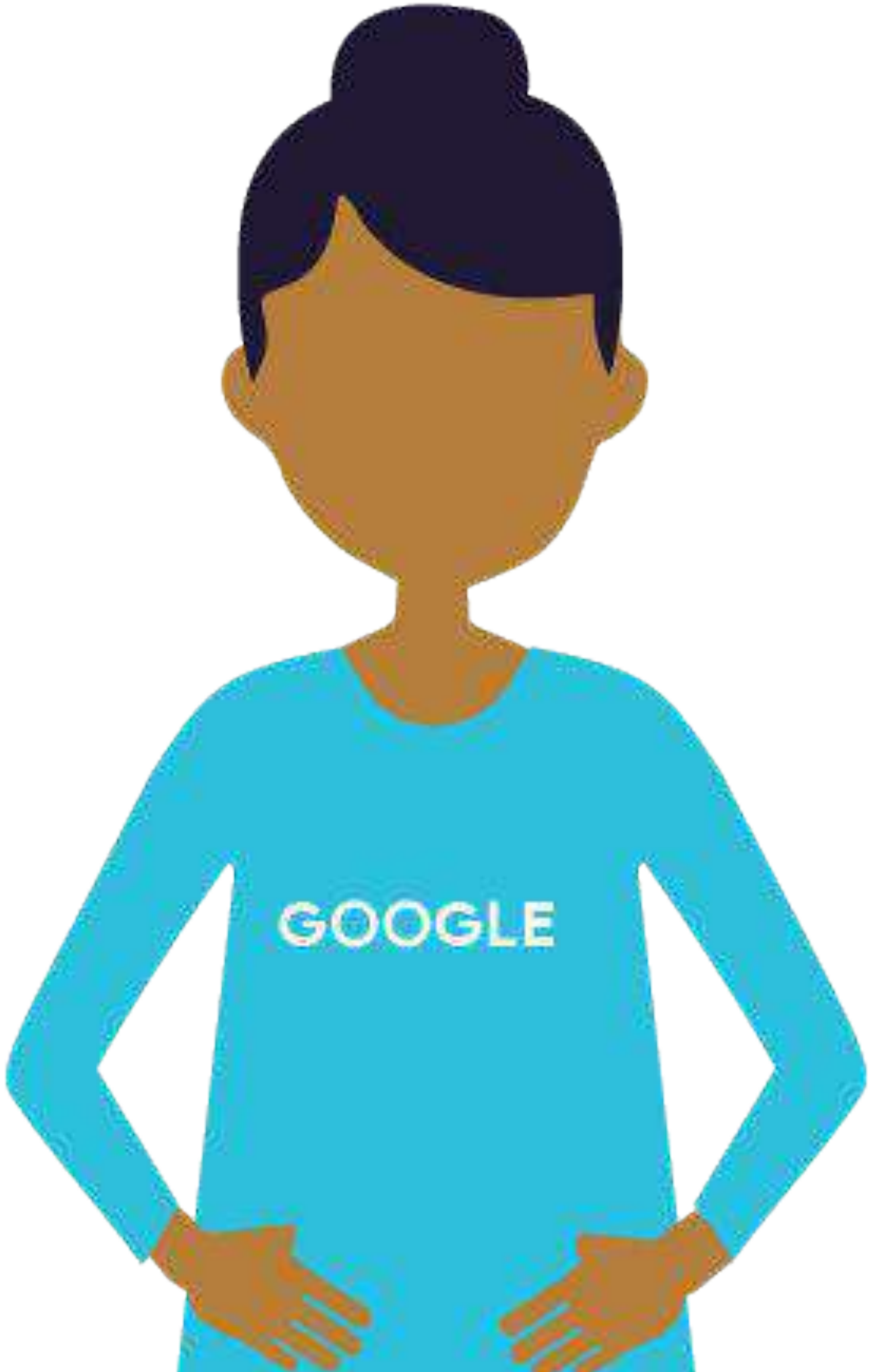
Cloud Machine Learning Engine

For Batch Pipelines

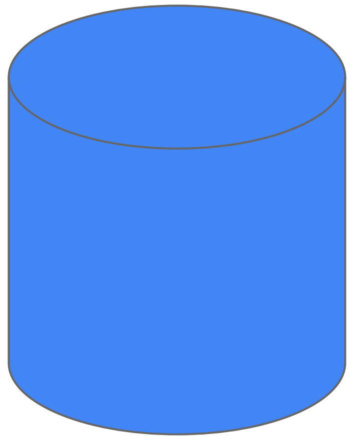



# Implementation Options

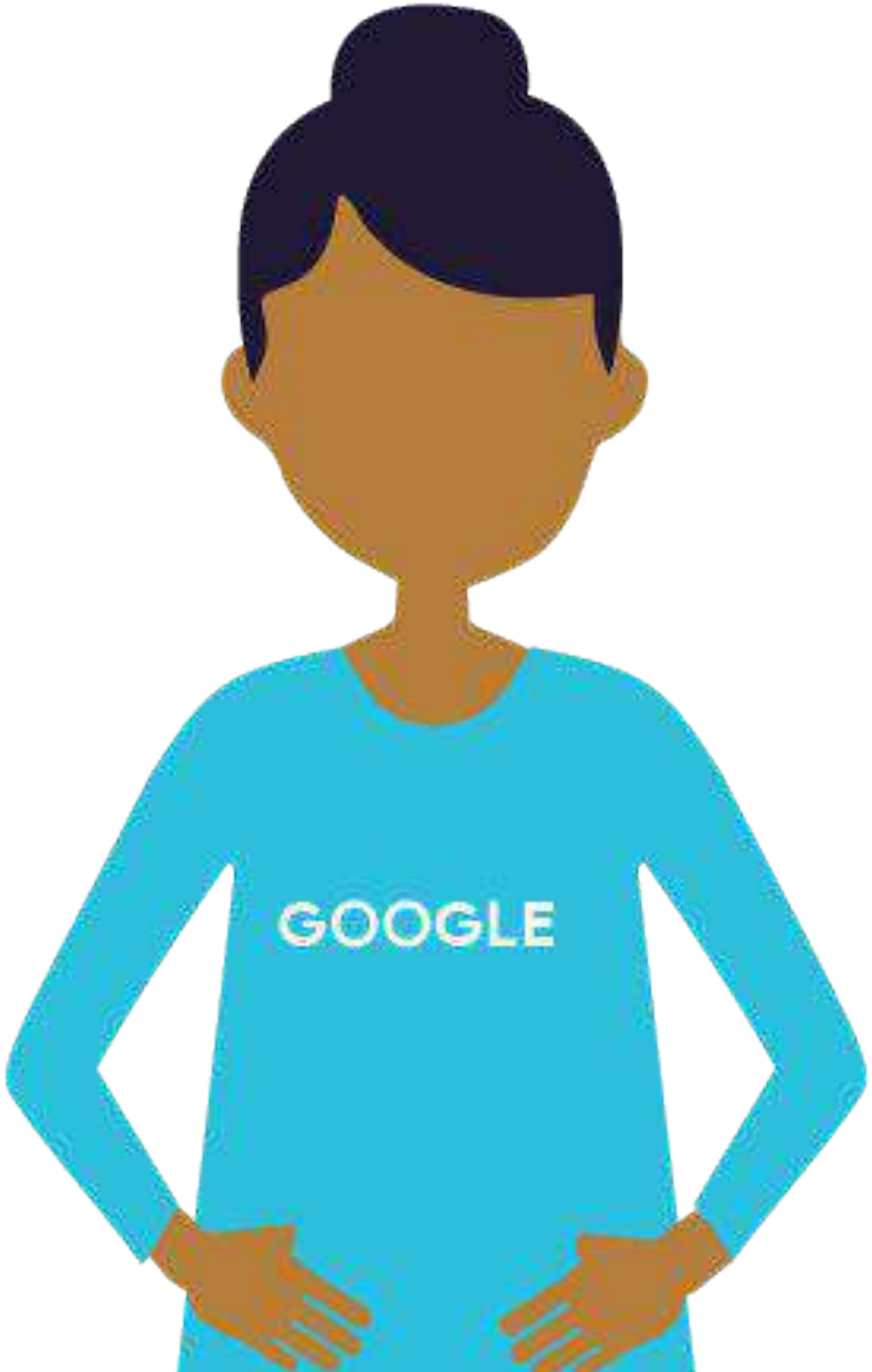
| REST/HTTP API | Cloud Machine Learning Engine | Cloud Dataflow |
|---|---|---|
| For Streaming Pipelines | For Batch Pipelines | For Batch and Streaming Pipelines |

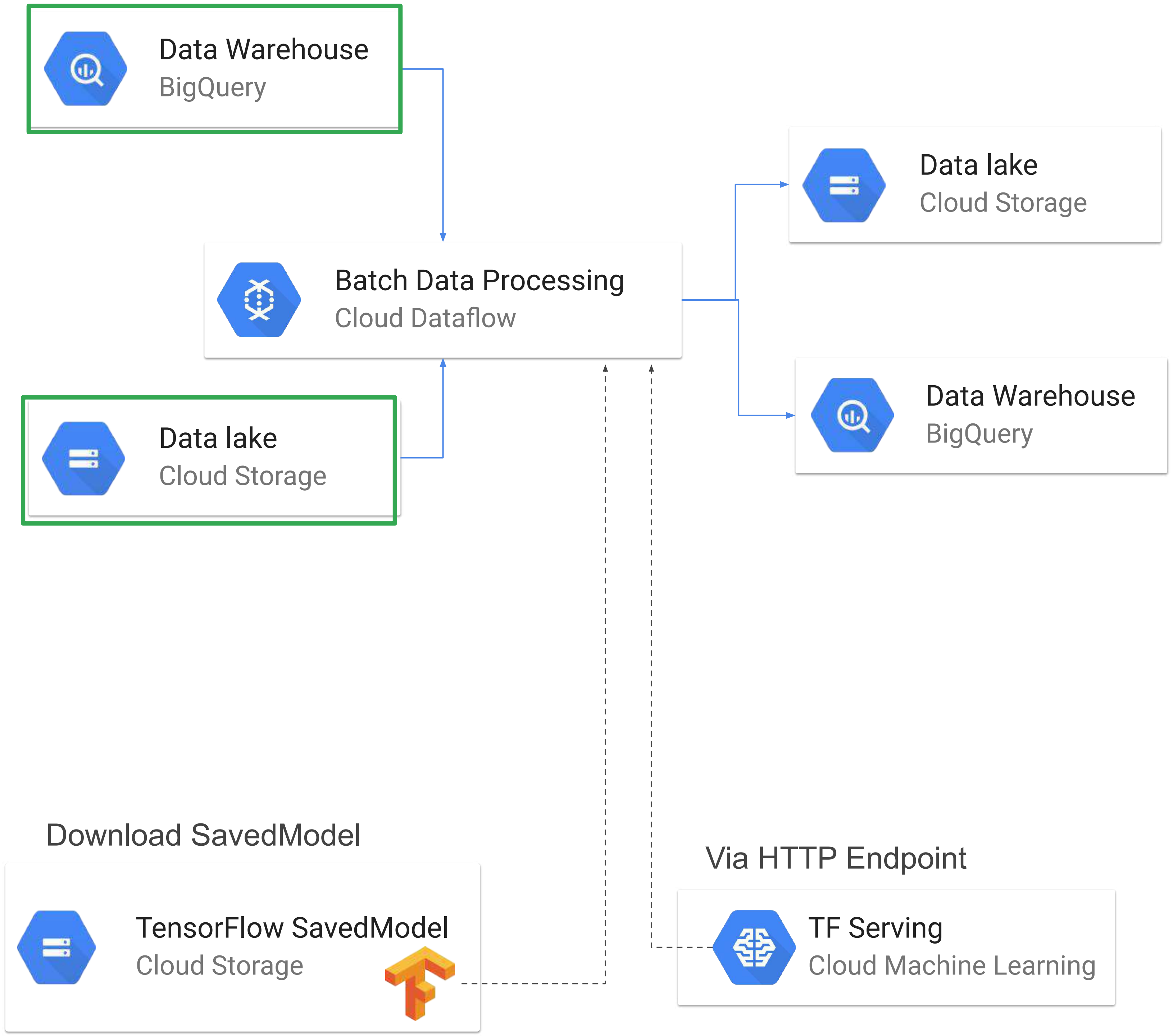


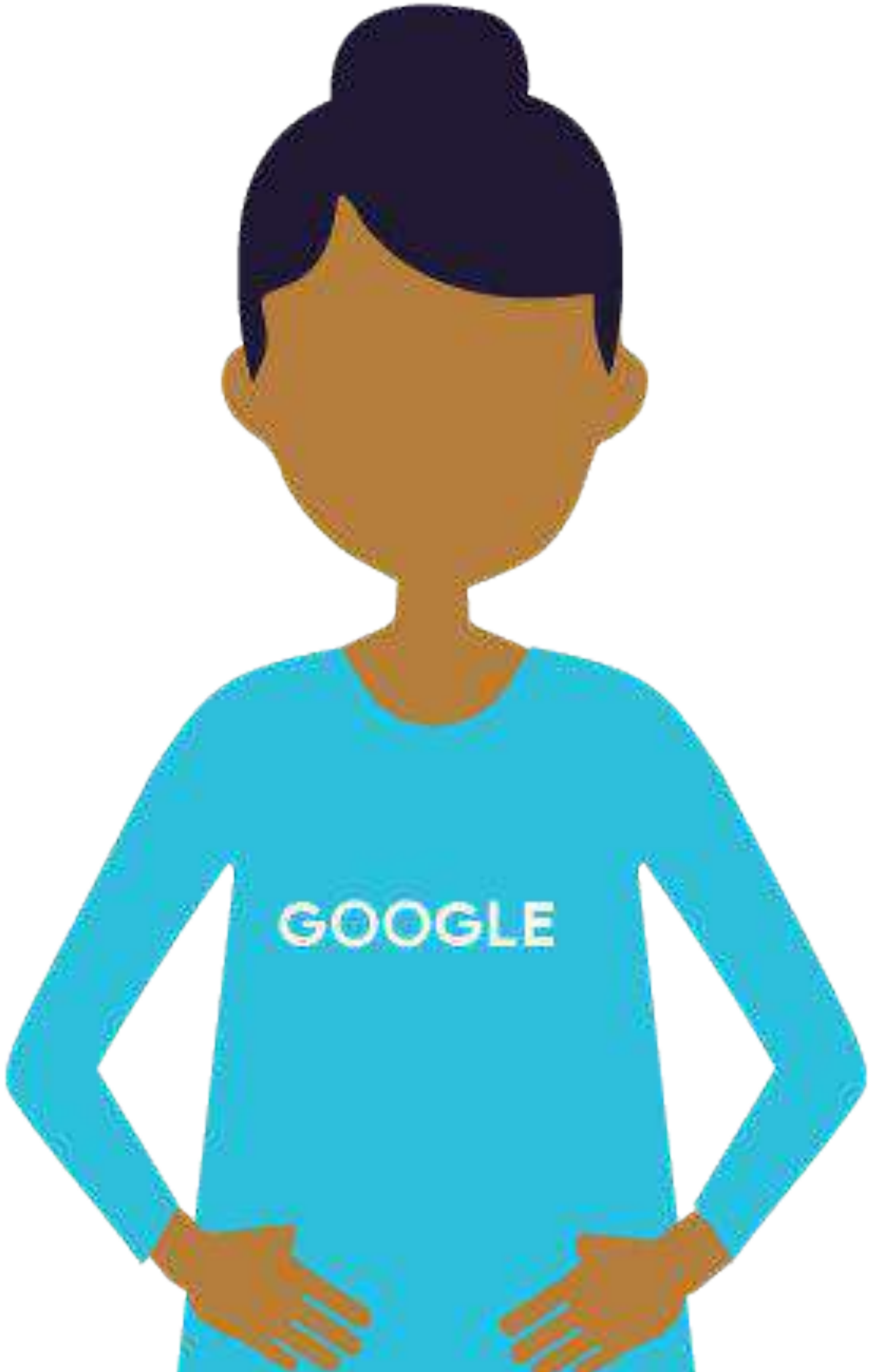

# Batch = Bounded Dataset

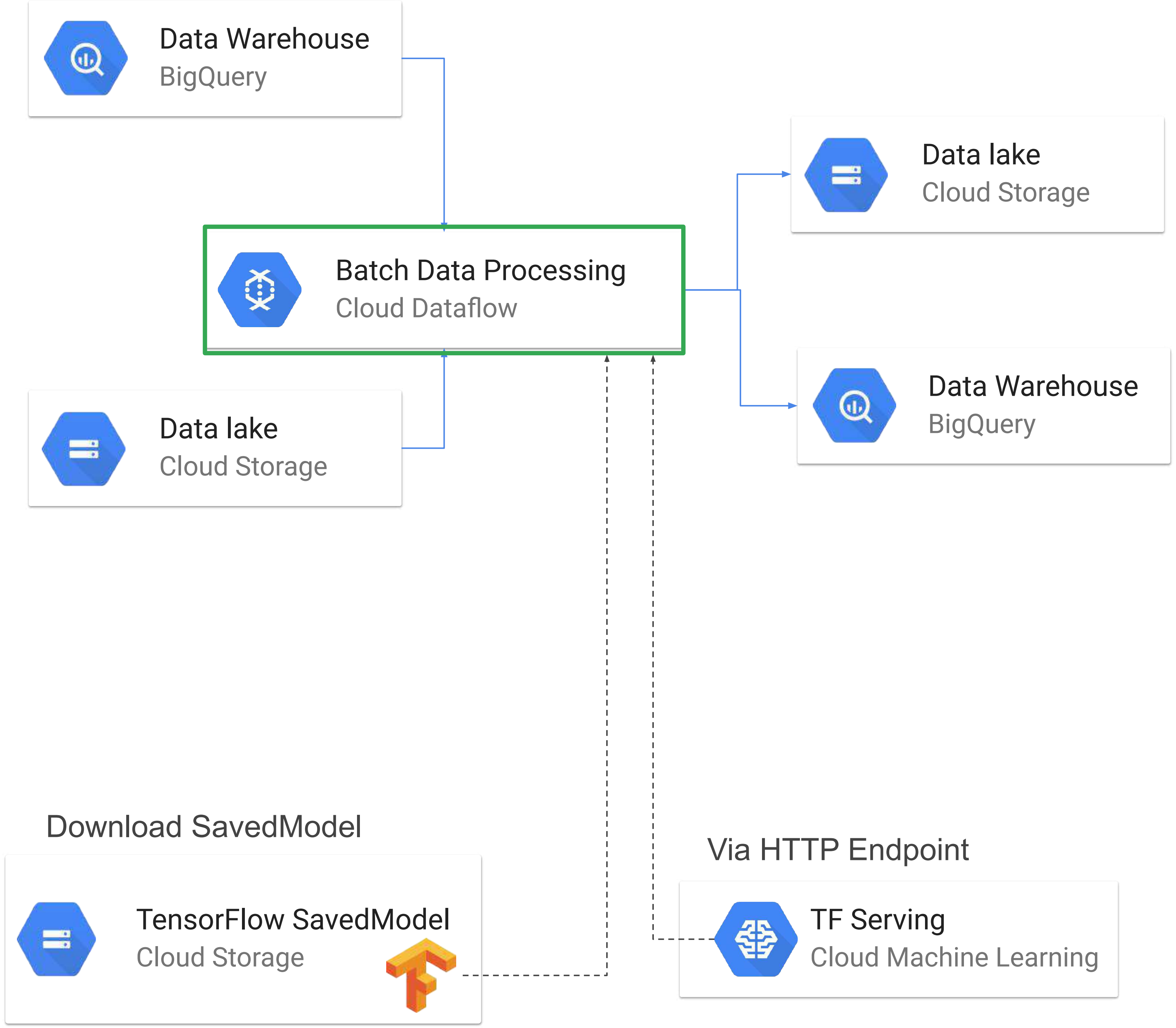


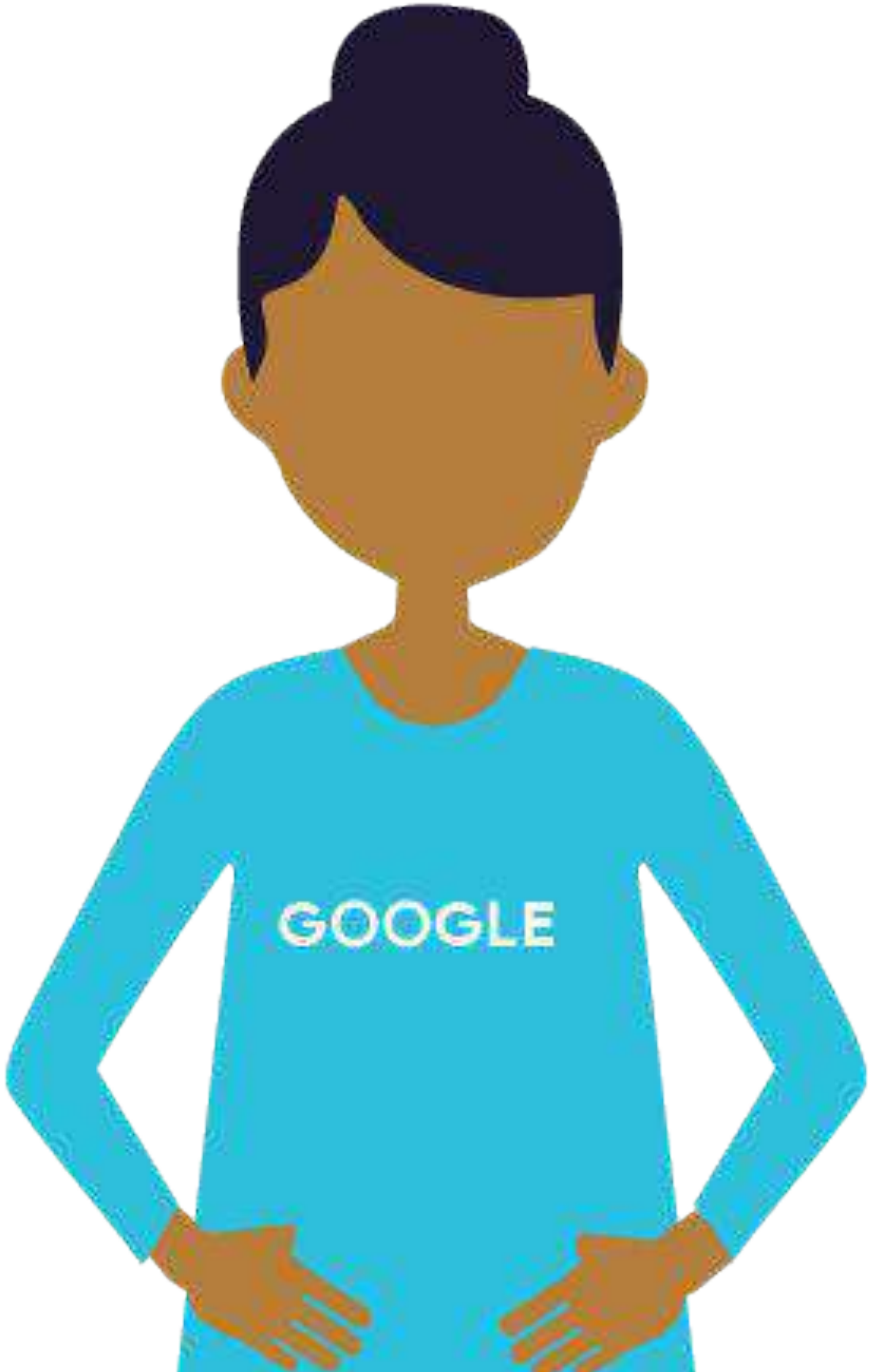
```
SELECT * FROM sales
WHERE date = '2018-01-01'
```

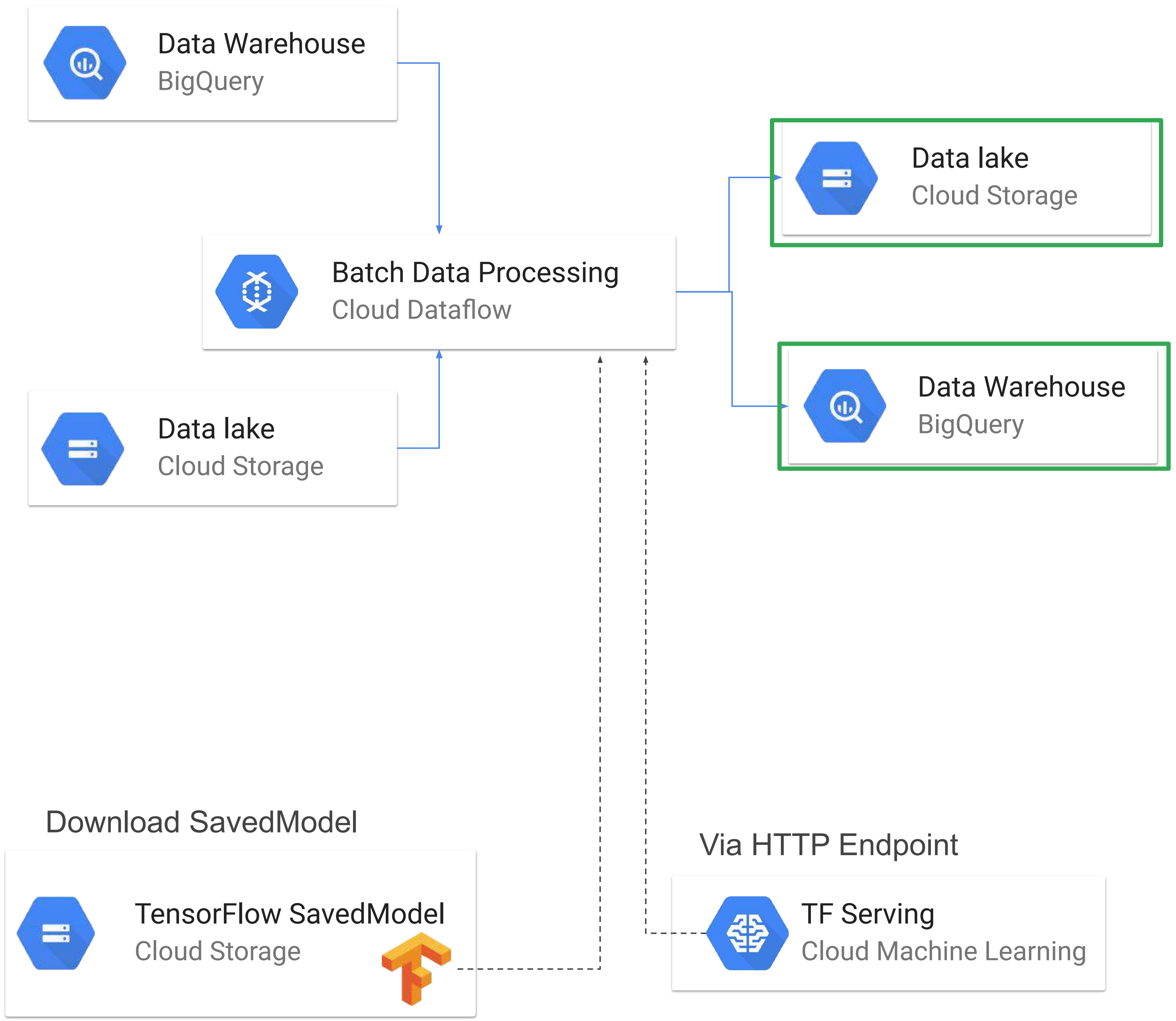

GOOGLE

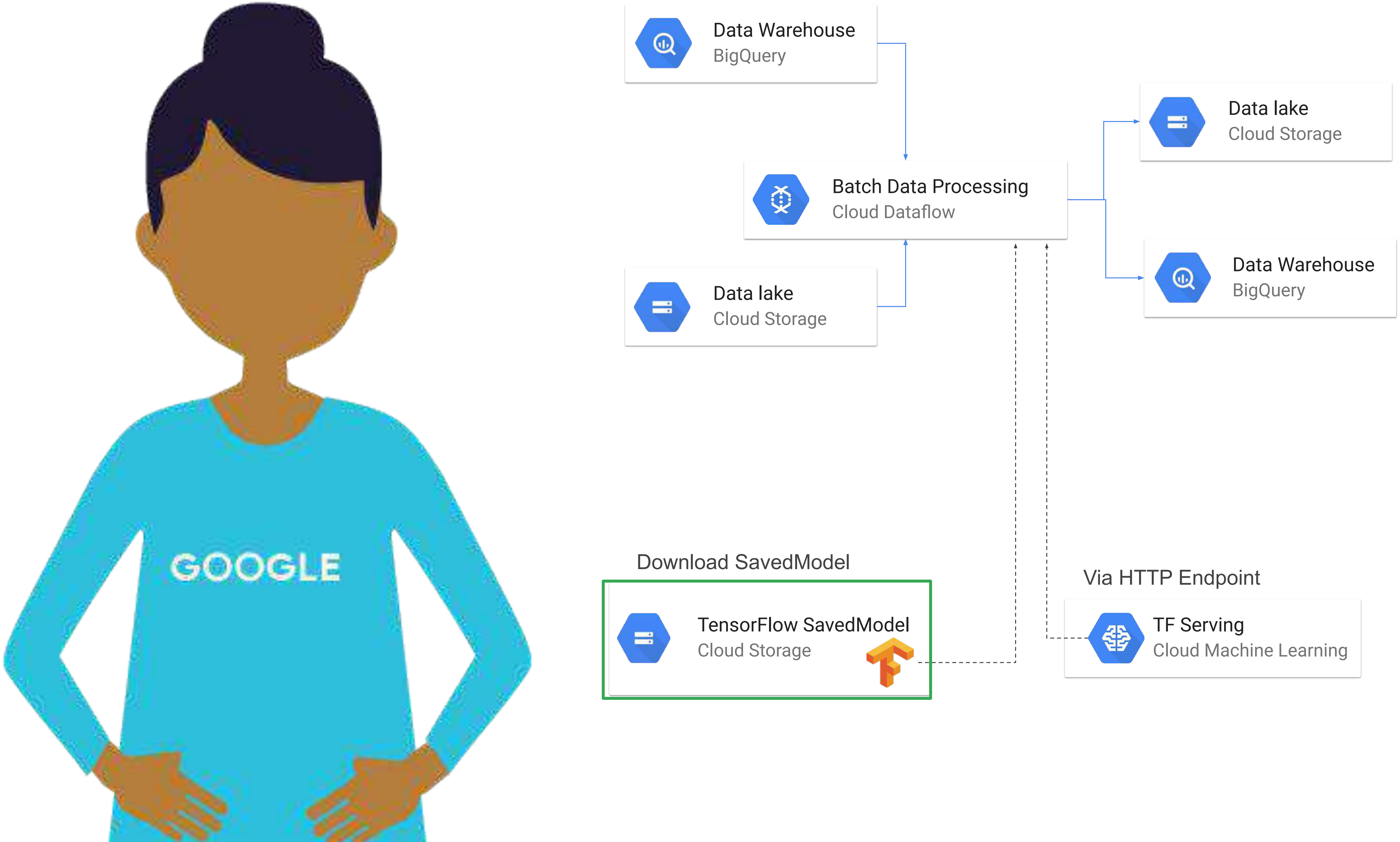

Data Warehouse
BigQuery
Batch Data Processing
Cloud Dataflow
Data lake
Cloud Storage
Data lake
Cloud Storage
Data Warehouse
BigQuery
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning

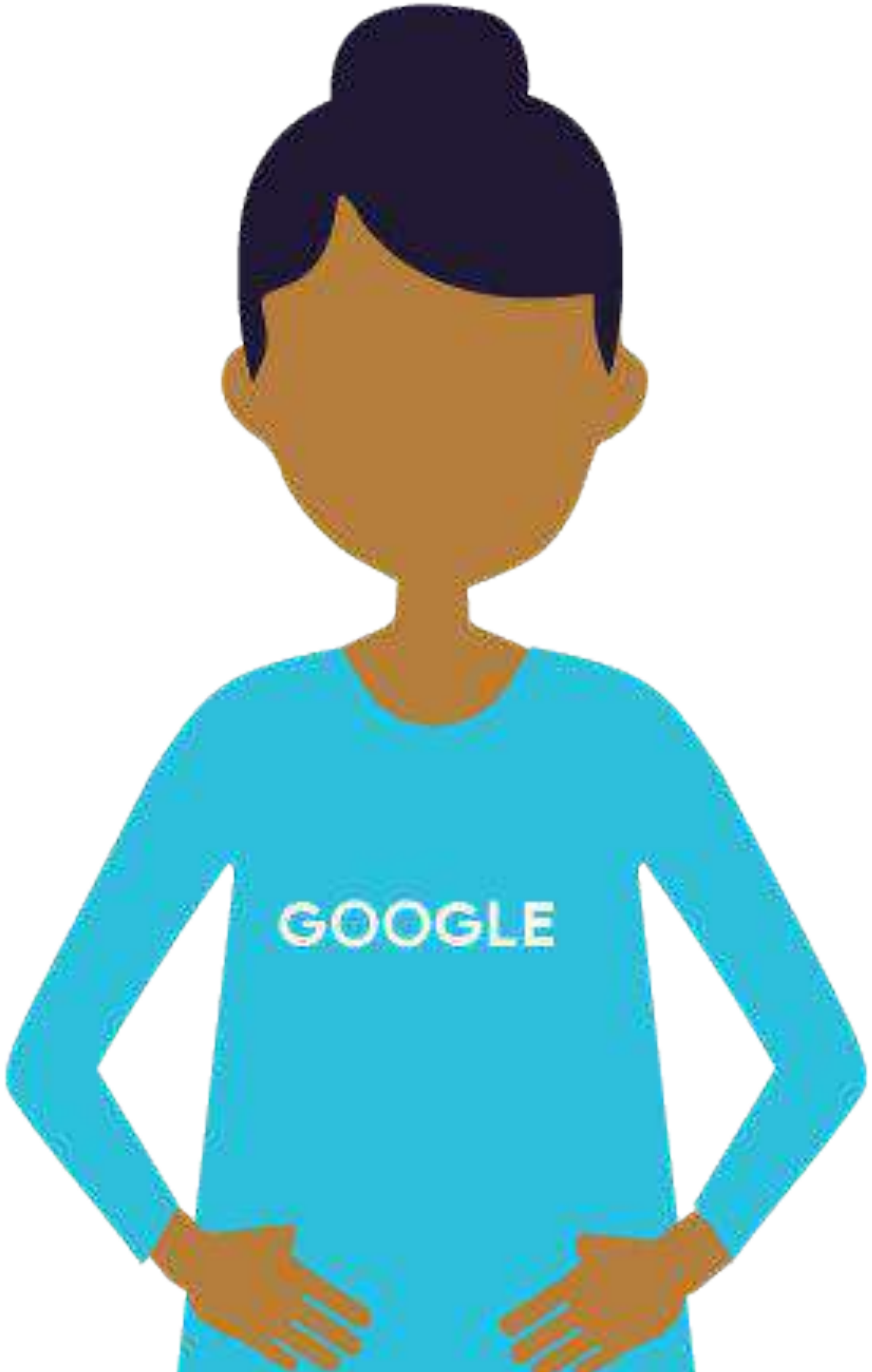

GOOGLE

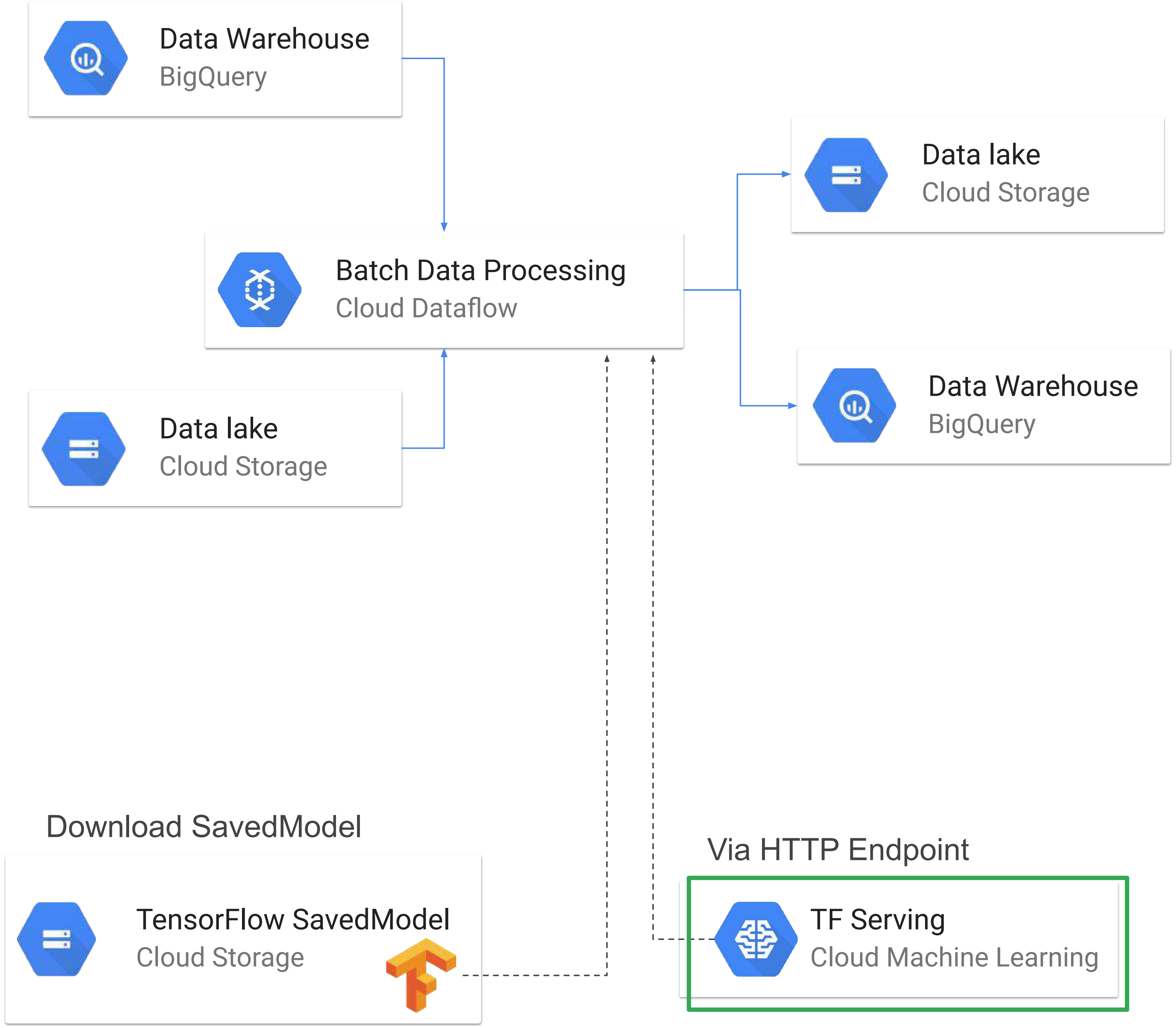

Data Warehouse
BigQuery
Batch Data Processing
Cloud Dataflow
Data lake
Cloud Storage
Data lake
Cloud Storage
Data Warehouse
BigQuery
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning


GOOGLE

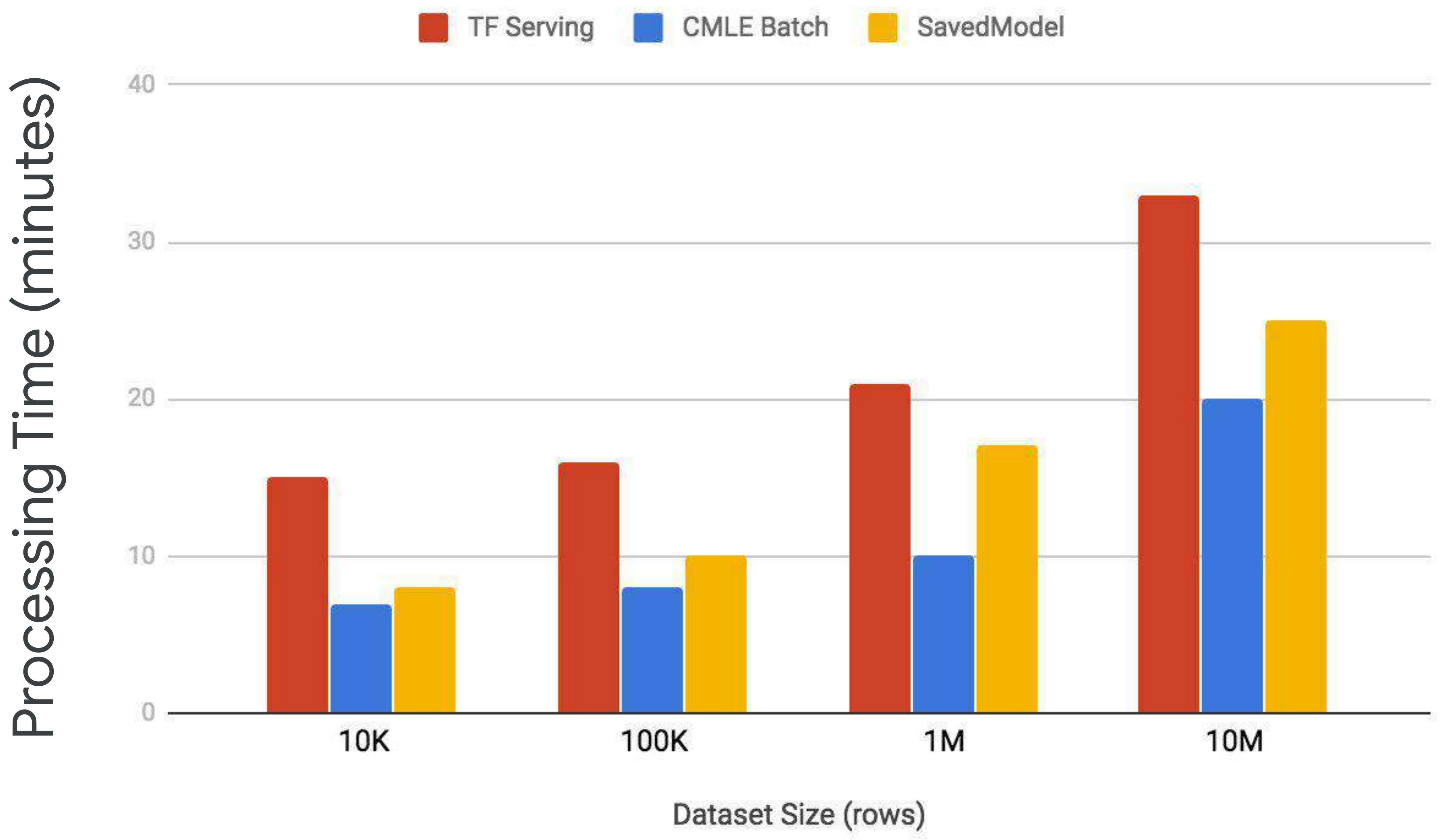

Data Warehouse
BigQuery
Batch Data Processing
Cloud Dataflow
Data lake
Cloud Storage
Data lake
Cloud Storage
Data Warehouse
BigQuery
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning

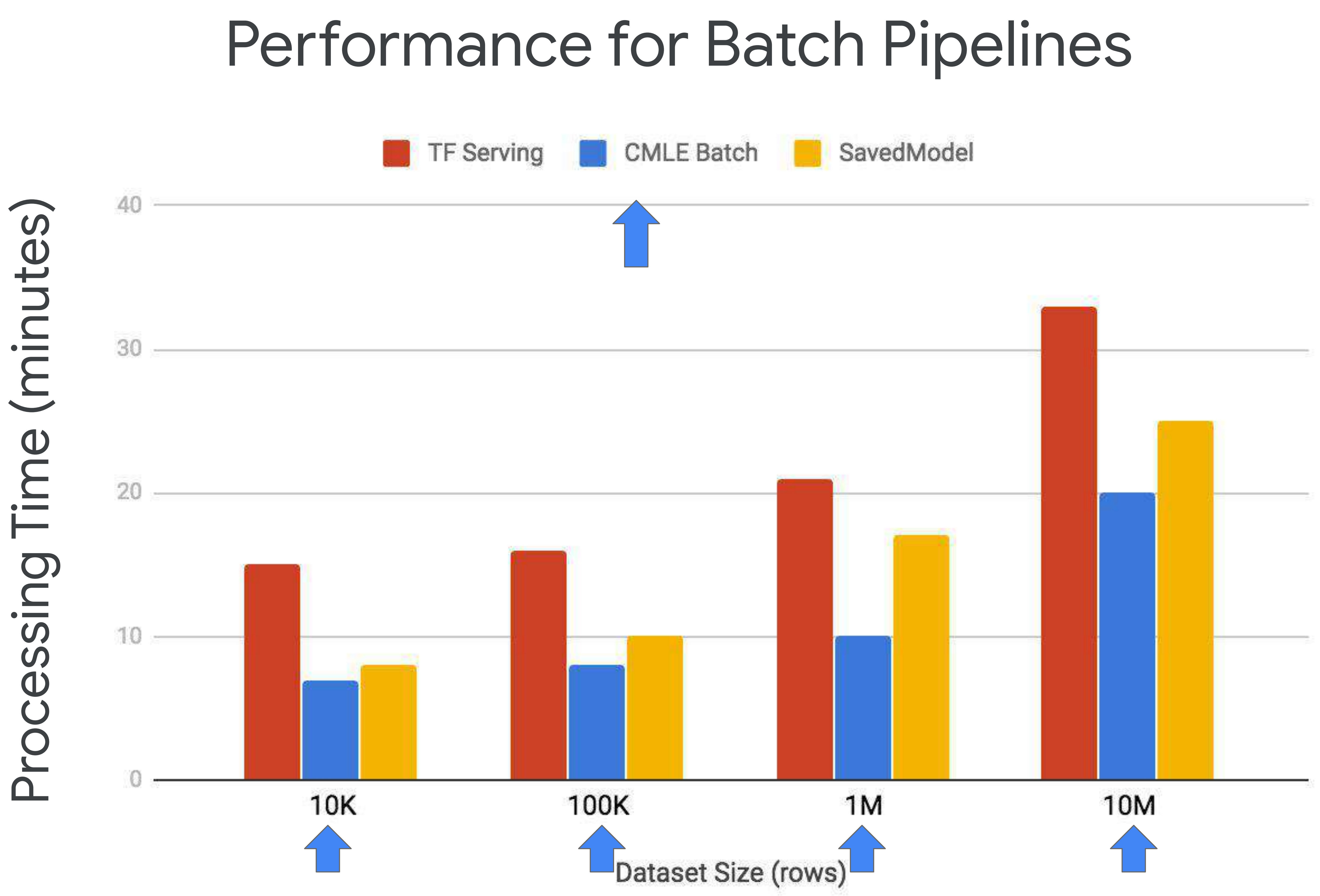

GOOGLE
Data Warehouse
BigQuery
Data lake
Cloud Storage
Batch Data Processing
Cloud Dataflow
Data lake
Cloud Storage
Data Warehouse
BigQuery
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning


GOOGLE

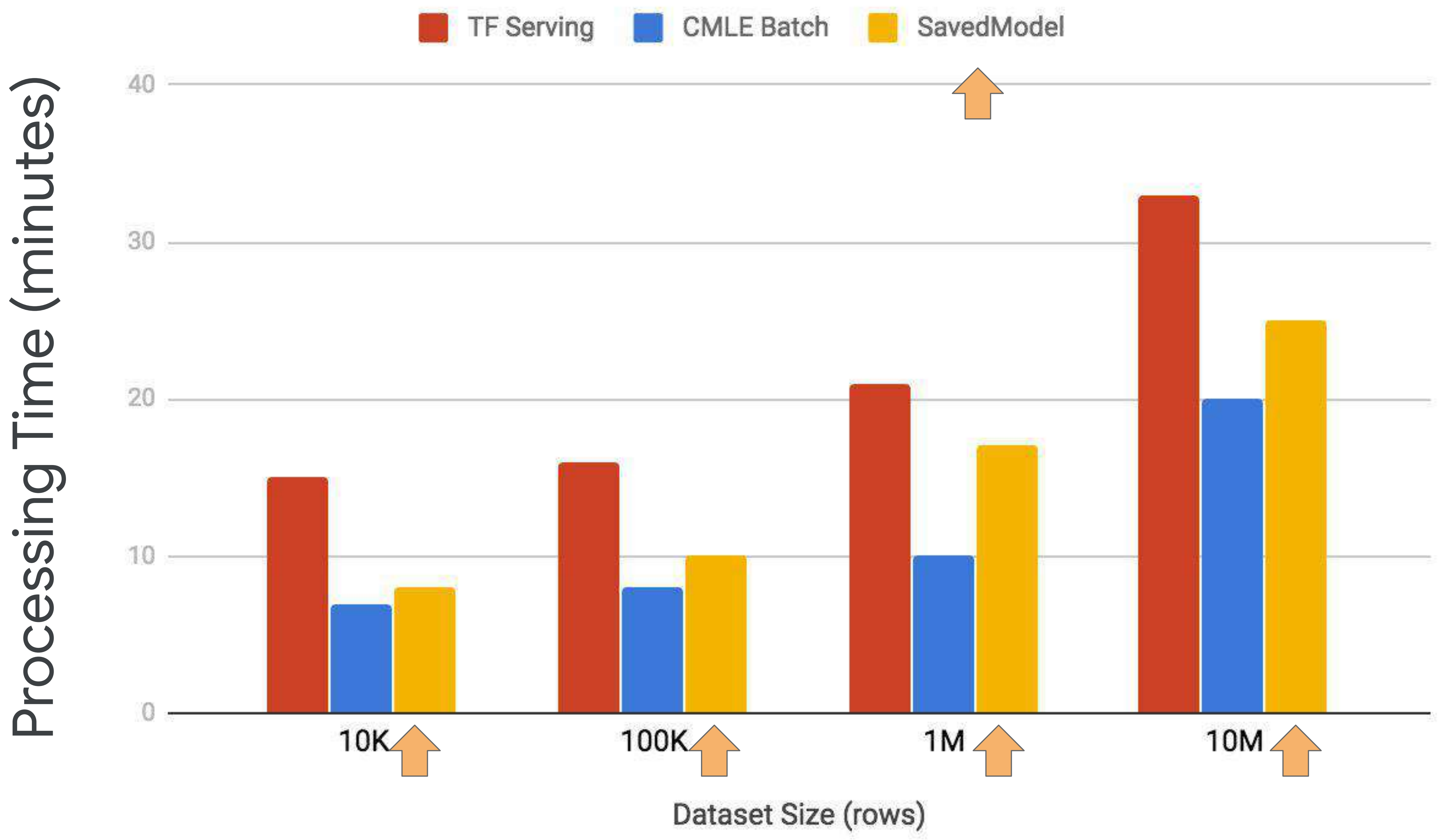

Data Warehouse
BigQuery
Batch Data Processing
Cloud Dataflow
Data lake
Cloud Storage
Data lake
Cloud Storage
Data Warehouse
BigQuery
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning

# Performance for Batch Pipelines

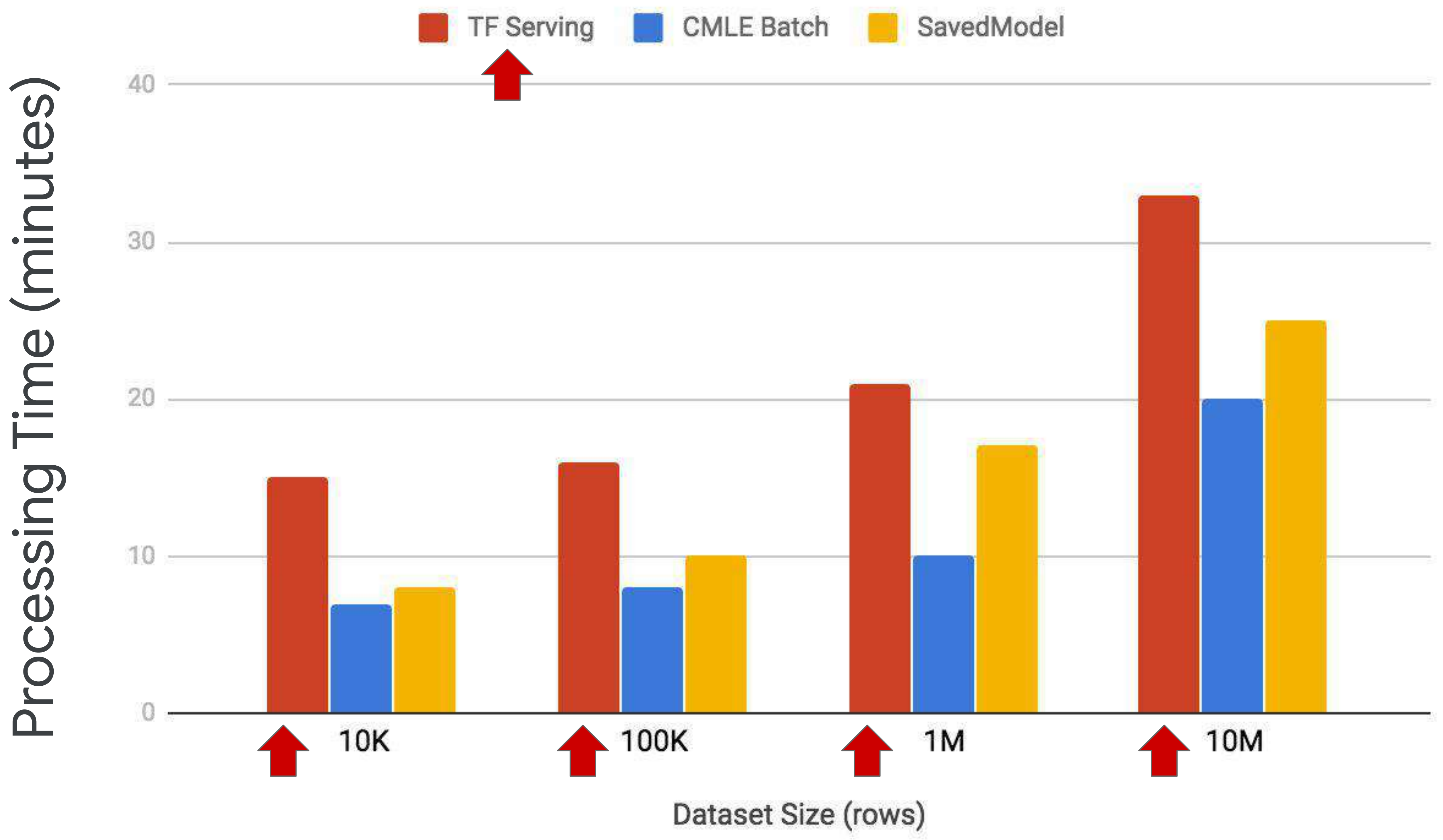



Performance for Batch Pipelines
TF Serving
CMLE Batch
SavedModel
Processing Time (minutes)
40
30
20
10
0
10K
100K
1M
10M
Dataset Size (rows)

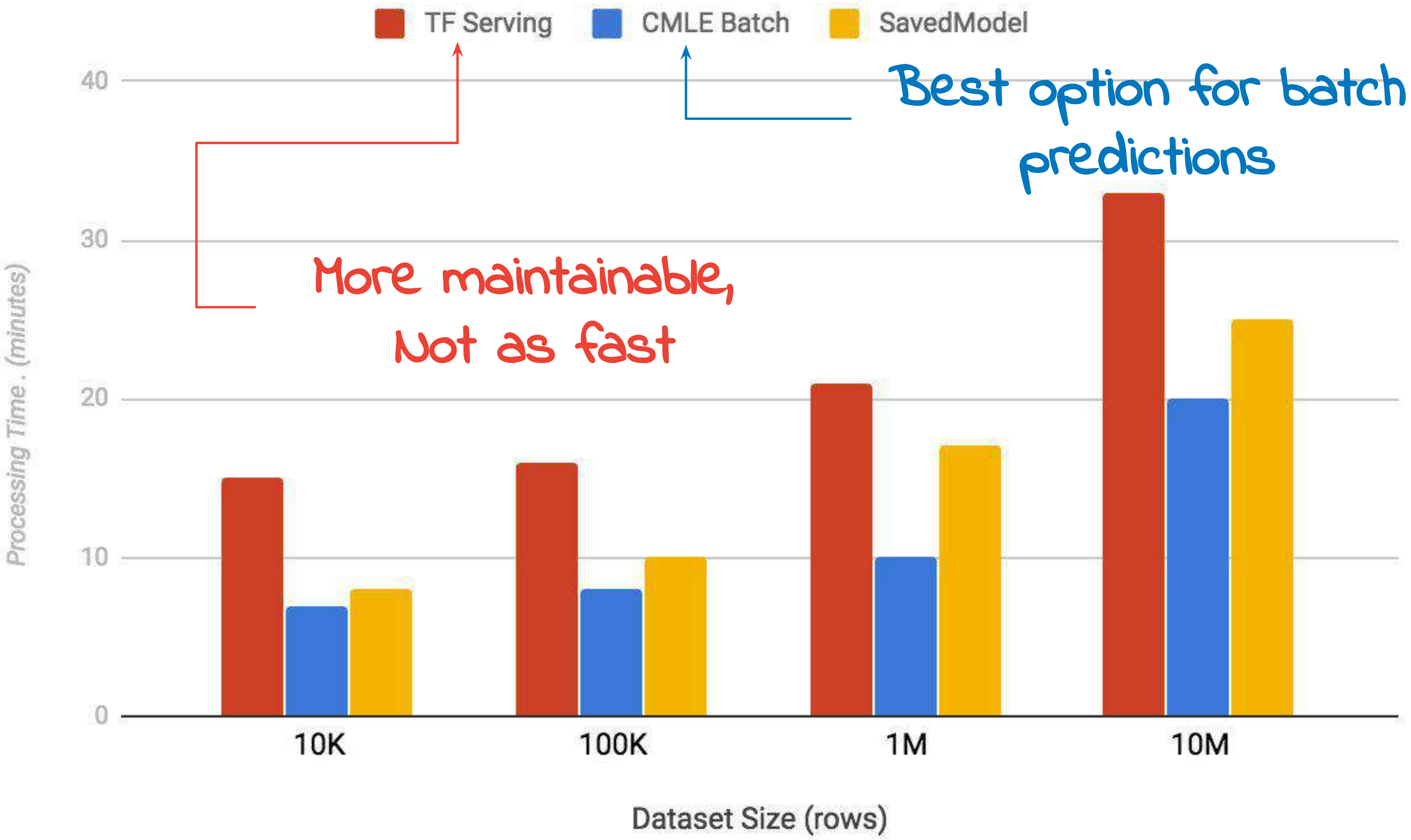
# Performance for Batch Pipelines

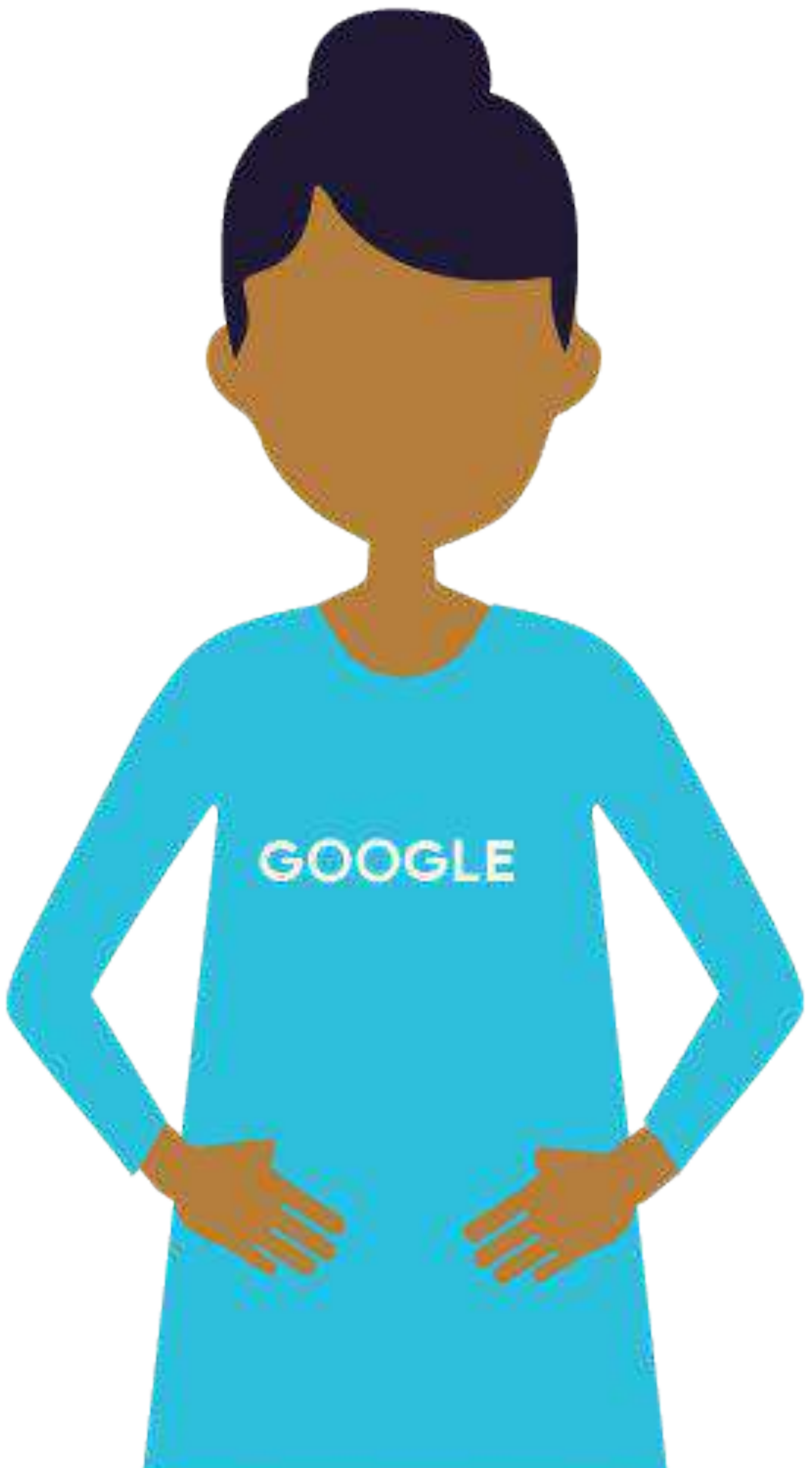


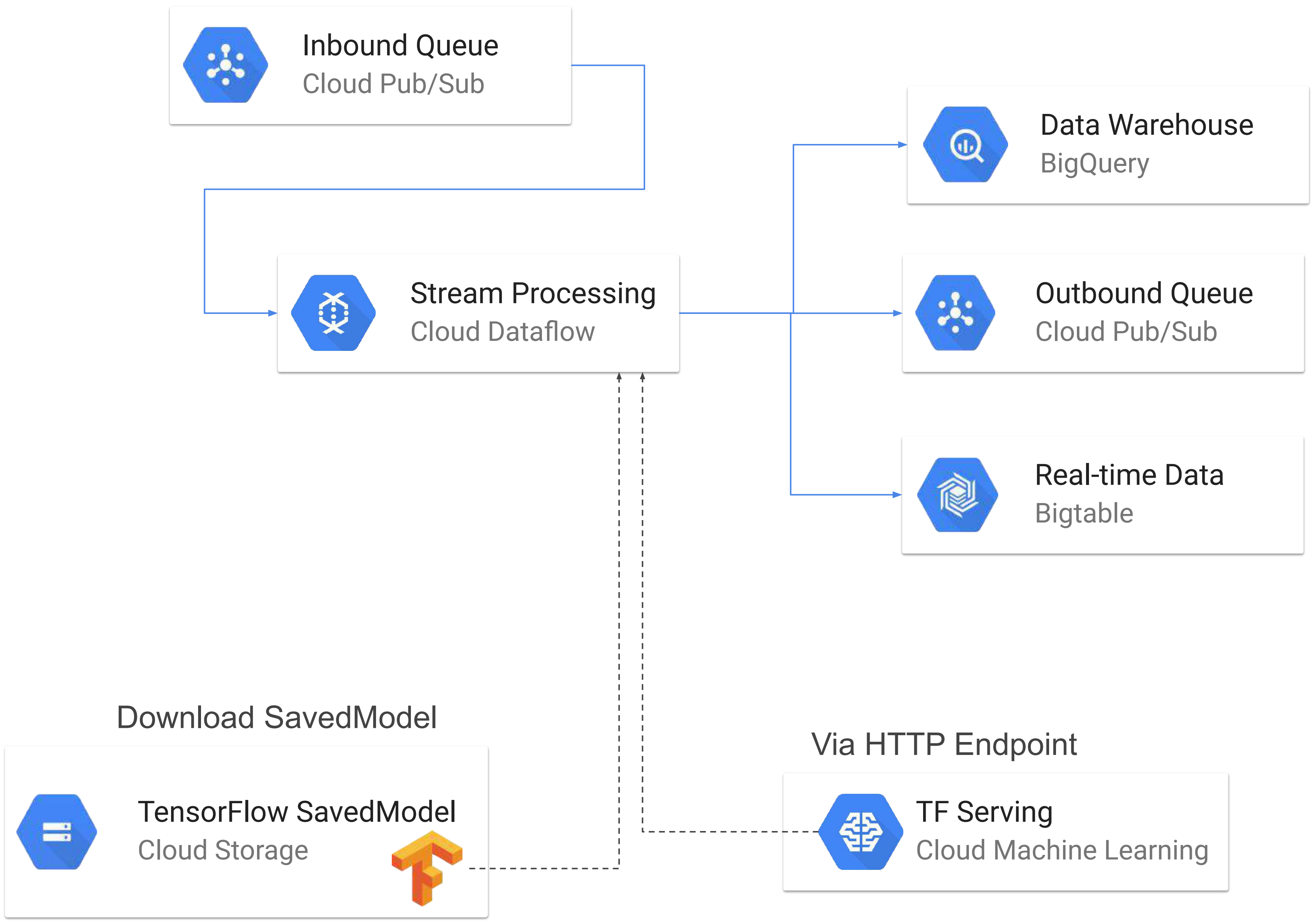
# Performance for Batch Pipelines



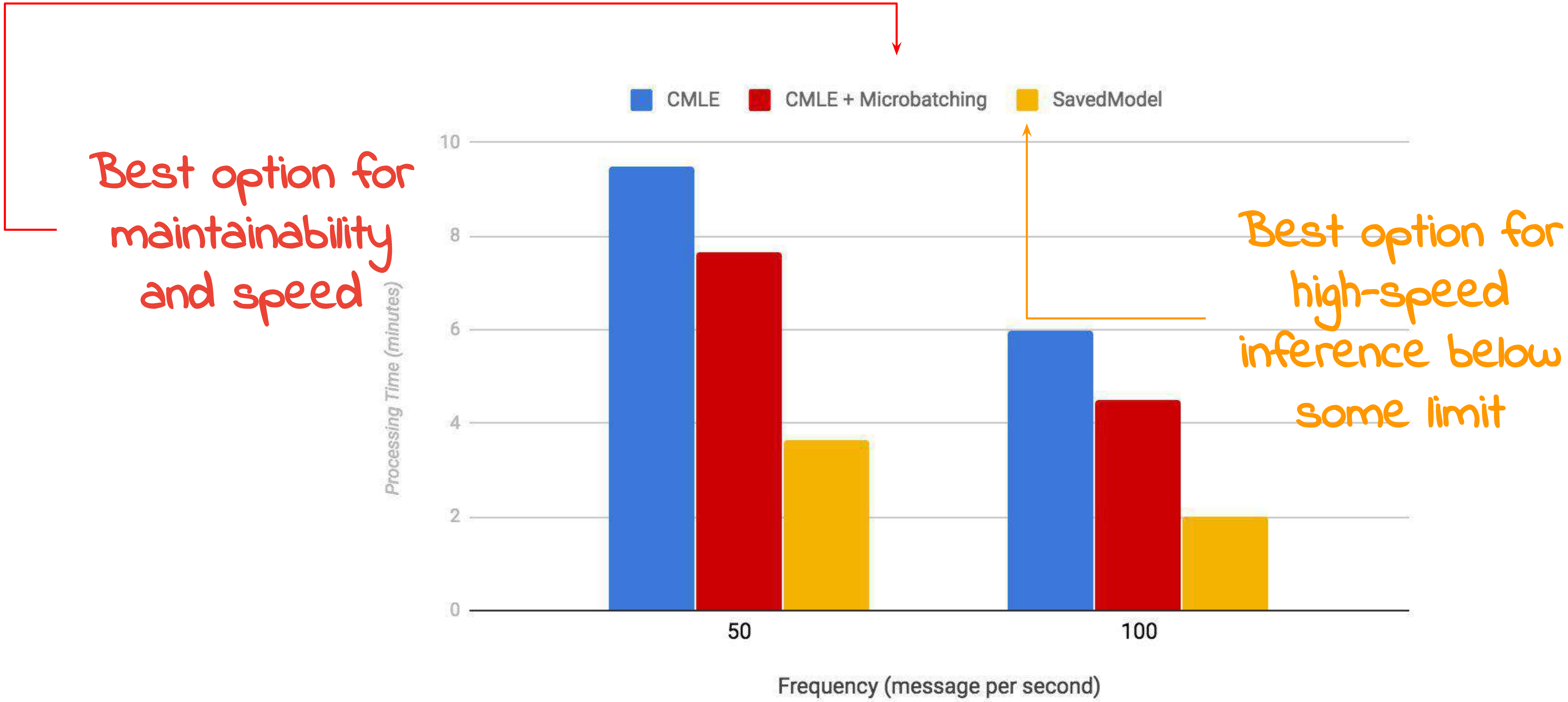
# Performance for Batch Pipelines




GOOGLE


Inbound Queue
Cloud Pub/Sub
Data Warehouse
BigQuery
Stream Processing
Cloud Dataflow
Outbound Queue
Cloud Pub/Sub
Real-time Data
Bigtable
Download SavedModel
TensorFlow SavedModel
Cloud Storage
Via HTTP Endpoint
TF Serving
Cloud Machine Learning

# Performance for Streaming Pipelines



Courses 7 - Production ML Systems

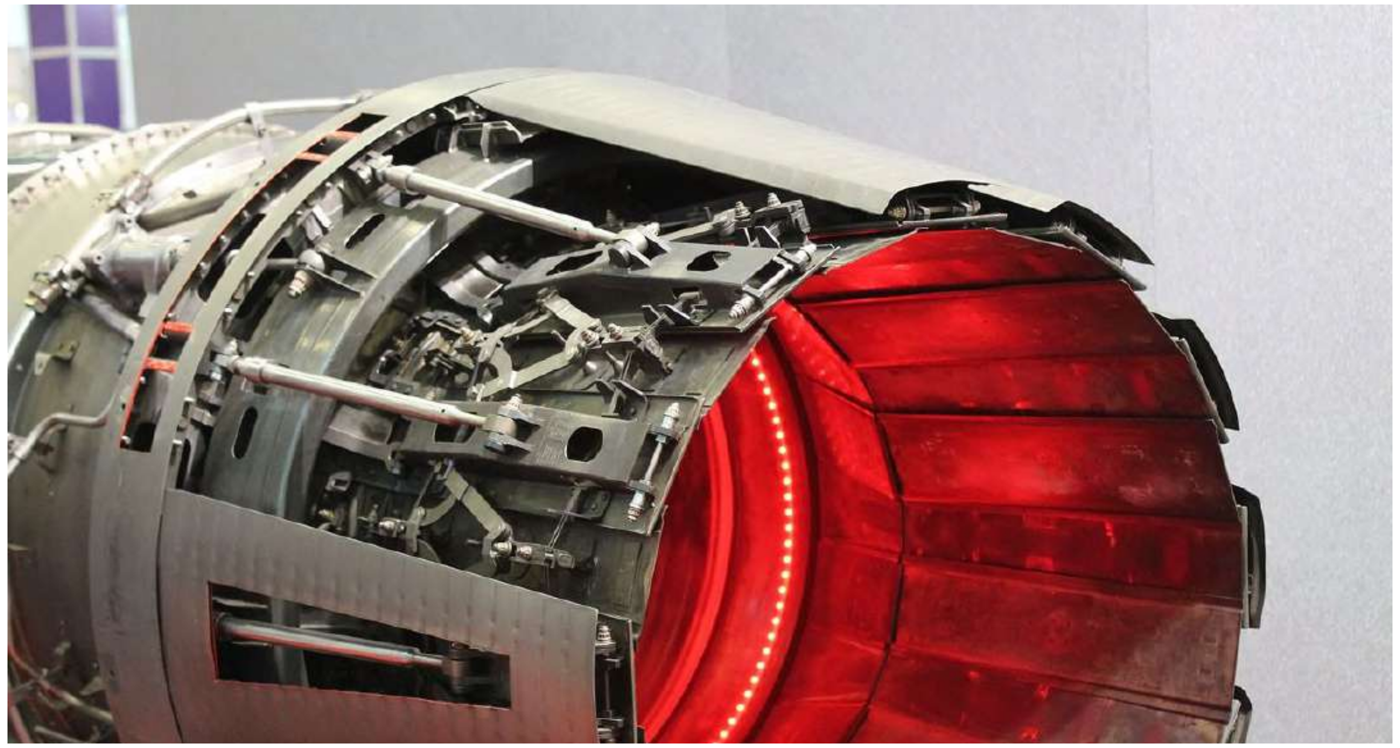
Module 4: Designing High-Performance ML Systems

Lesson Title: **Summary**

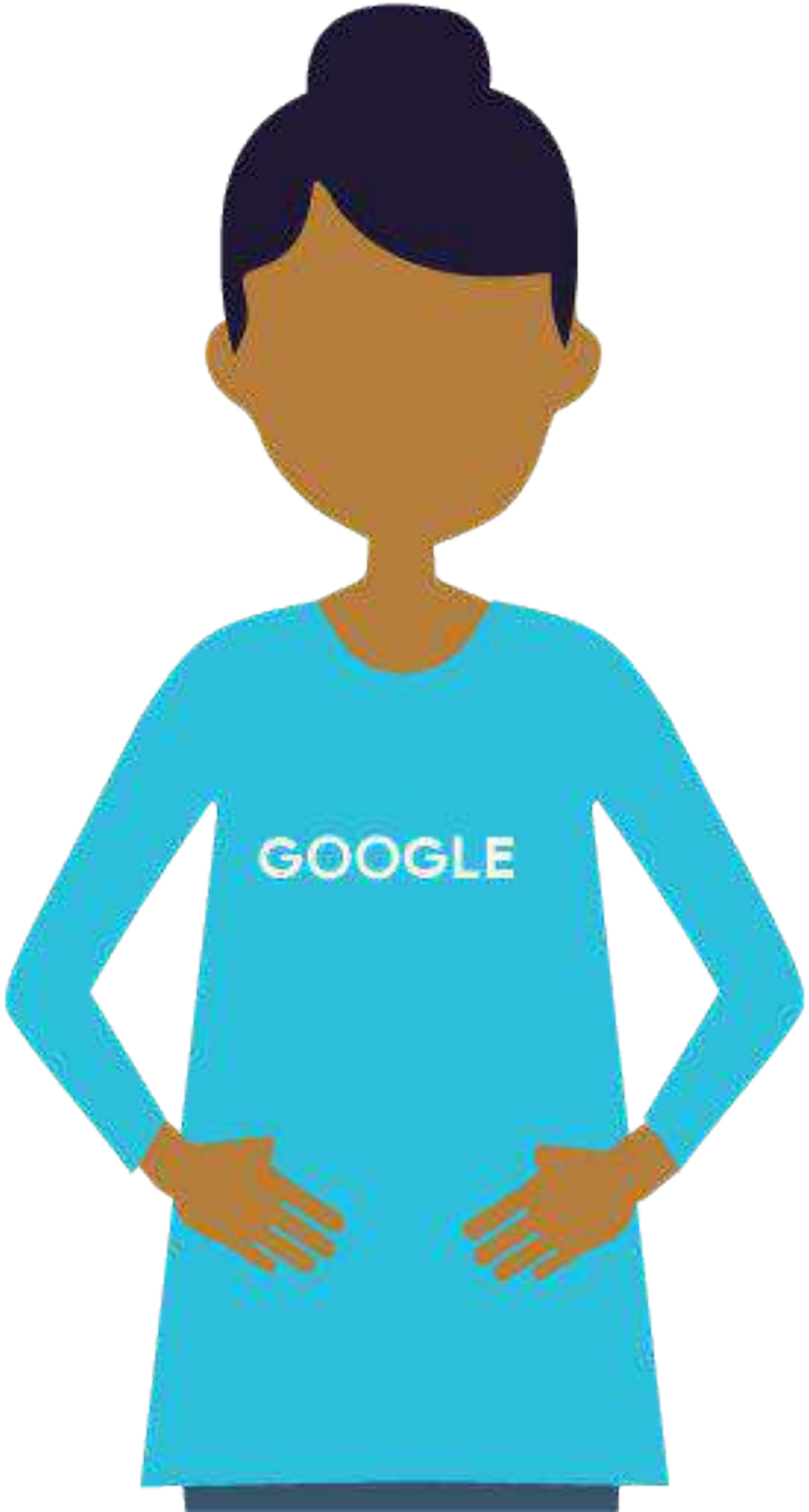
Format: Presenter

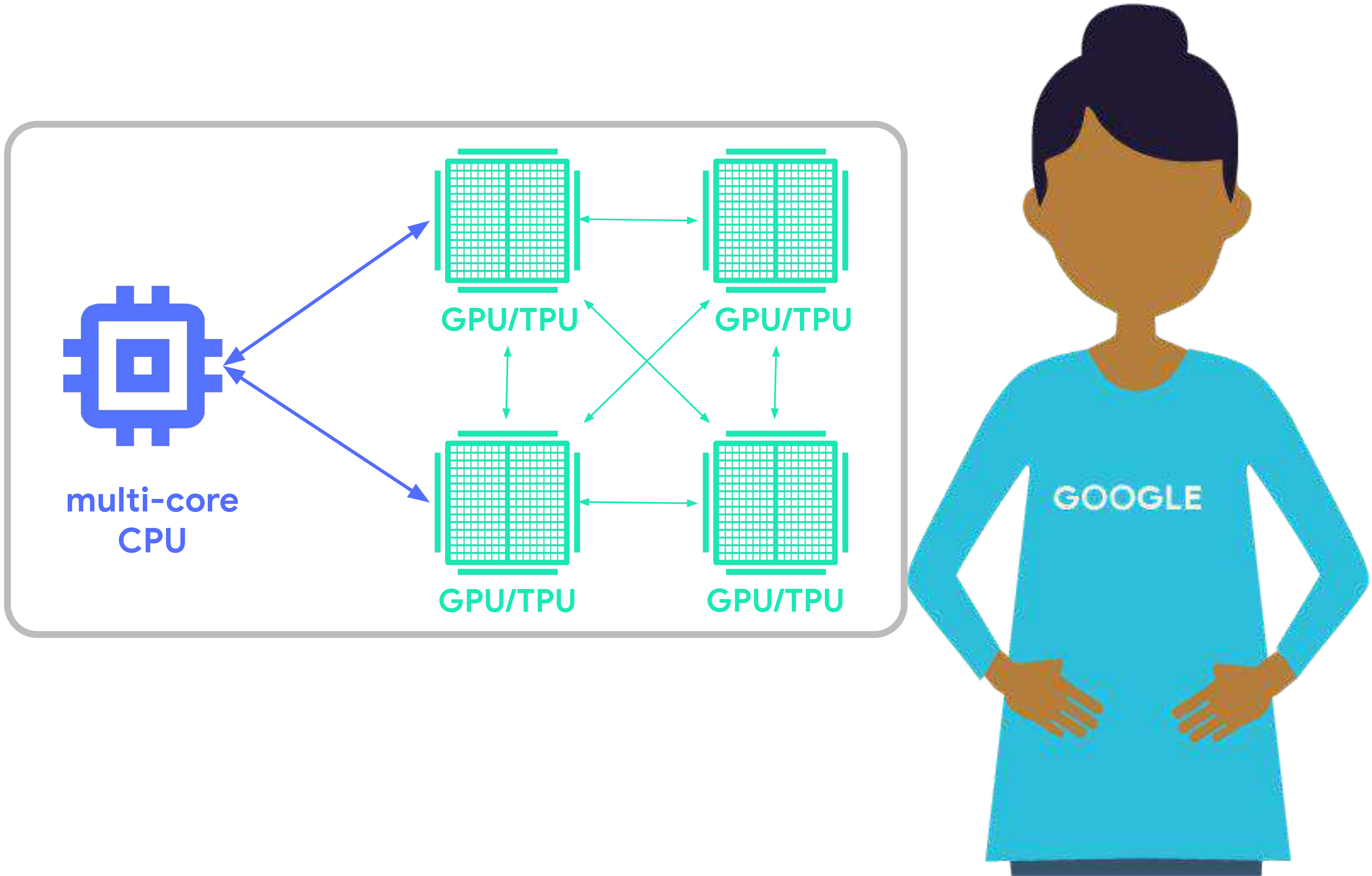
Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l13_summary

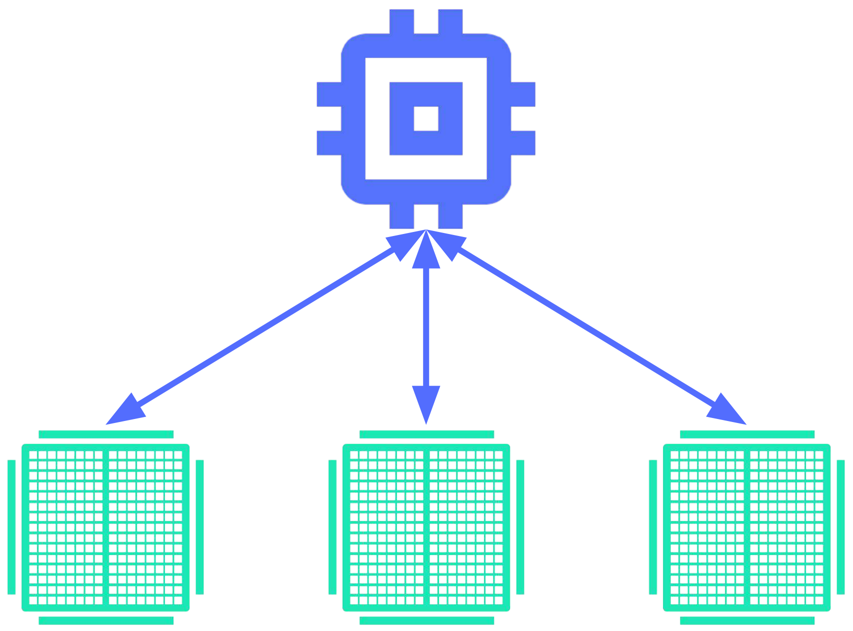



MIN
MAX
SPEED


GOOGLE


GPU/TPU
GPU/TPU
multi-core
CPU
GPU/TPU
GPU/TPU
GOOGLE

## Consider Async Parameter Server if...

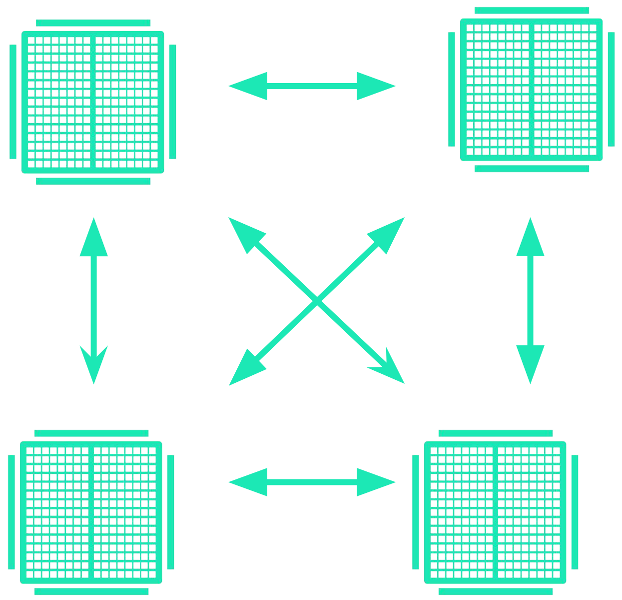


Many low-power or unreliable workers

More mature approach

Constrained by I/O

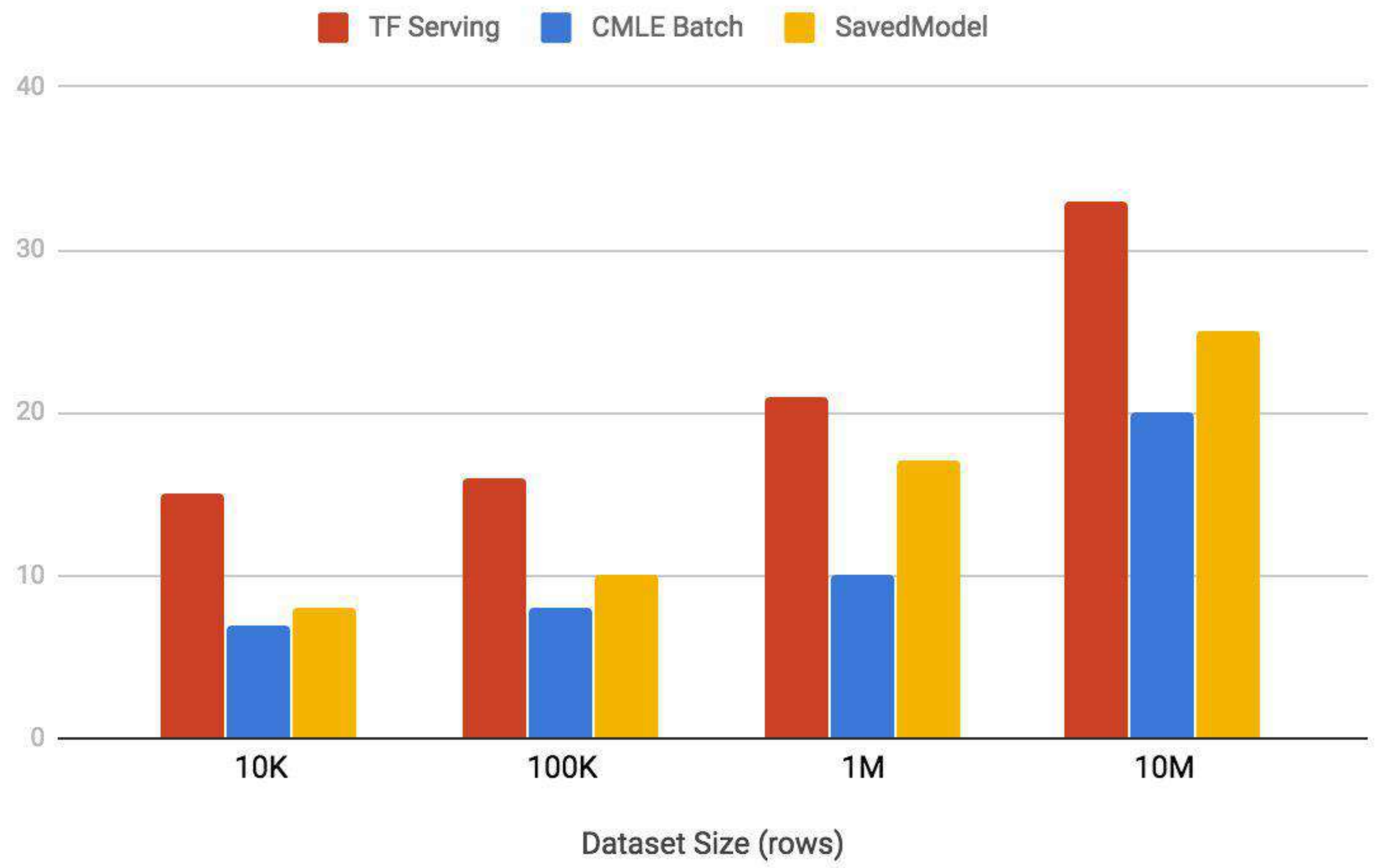
## Consider Sync Allreduce if...

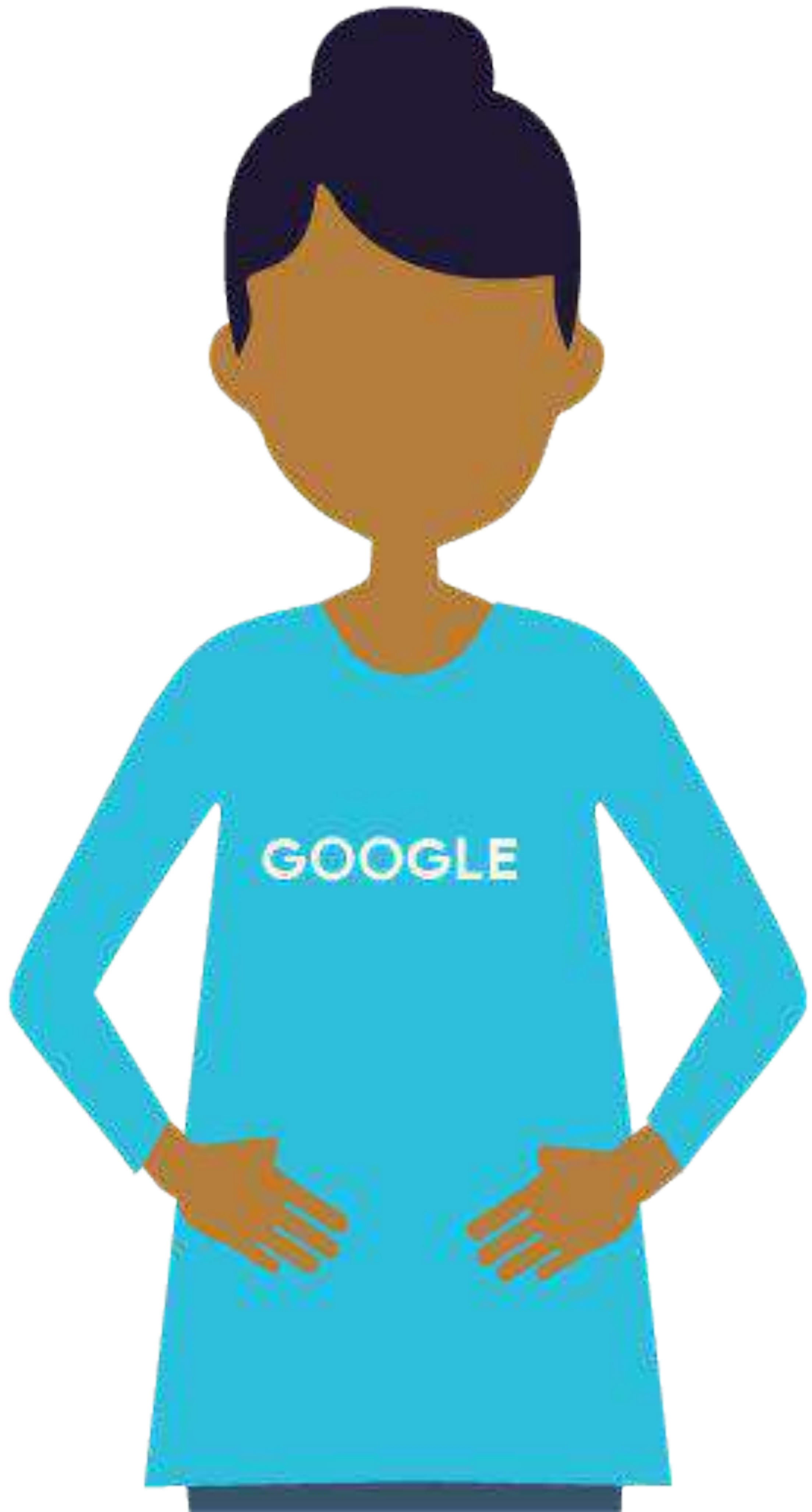


Multiple devices on one host
Fast devices with strong links (e.g. TPUs)

Better for multiple GPUs

Constrained by compute power


TF Serving
CMLE Batch
SavedModel
40
30
20
10
0
10K
100K
1M
10M
Dataset Size (rows)


GOOGLE

Courses 7 - Production ML Systems

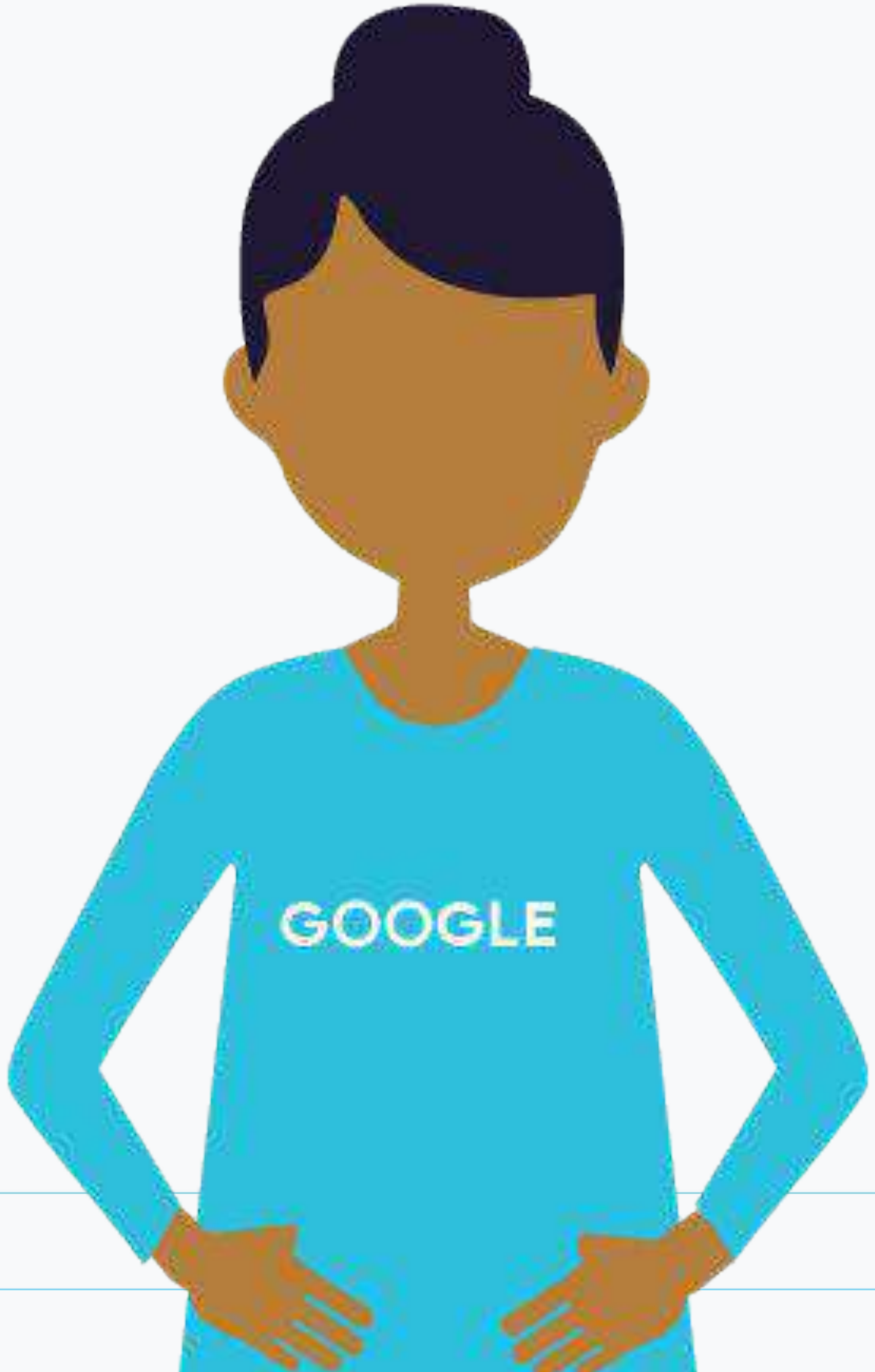
Module 4: Designing High-Performance ML Systems

Lesson Title: **Minimalist Core [optional]**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l14_minimalist_core_[optional]

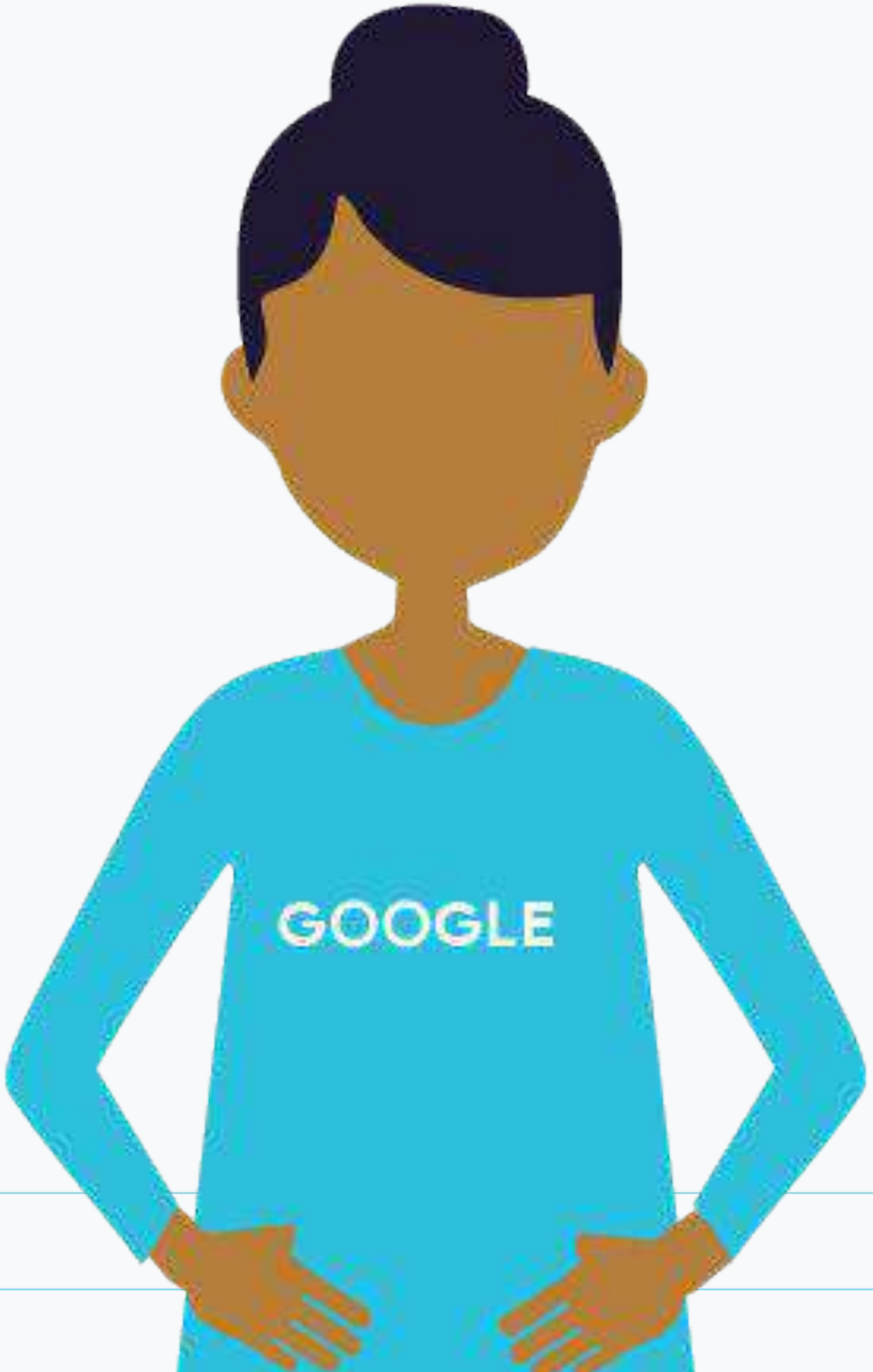




Title Safe >

< Action Safe

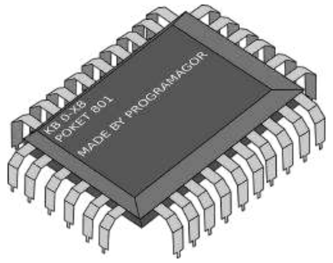
Distributed TensorFlow has a minimalist core. The core can be done on a single machine.




Client

/job:worker/task:0/

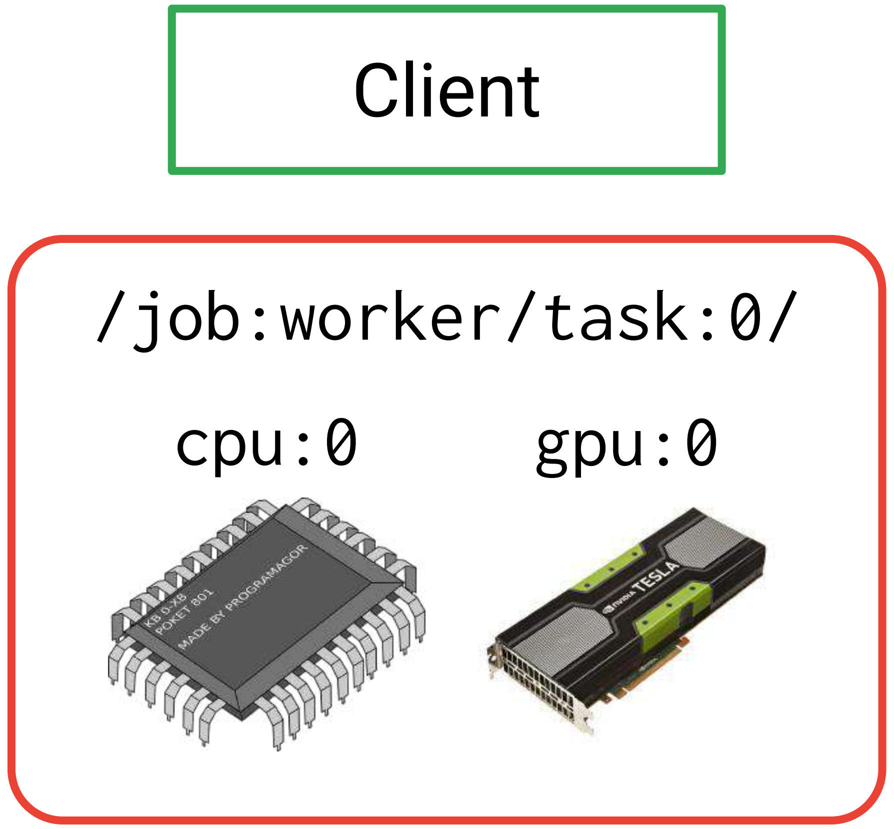
cpu:0

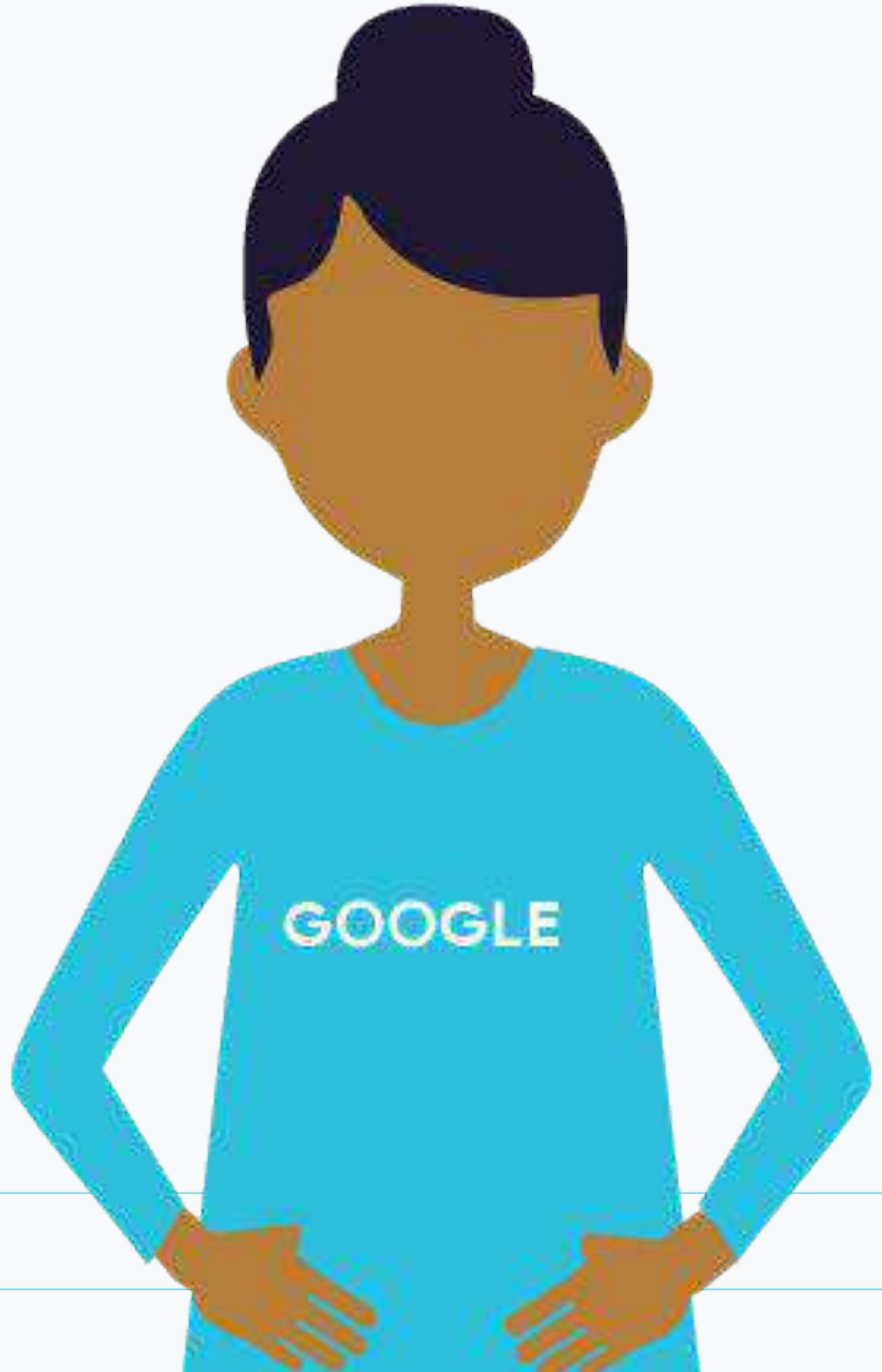


gpu:0

Distributed TensorFlow has a minimalist core

The core can be done on a single machine.





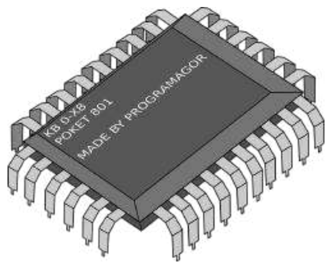

# You can assign variables to devices

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

/job:worker/task:0/

cpu:0



gpu:0

Title Safe >

< Action Safe

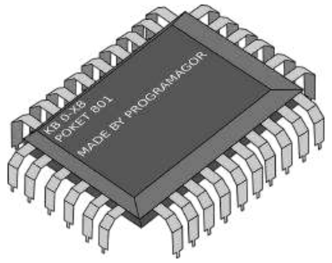
# You can assign variables to devices

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

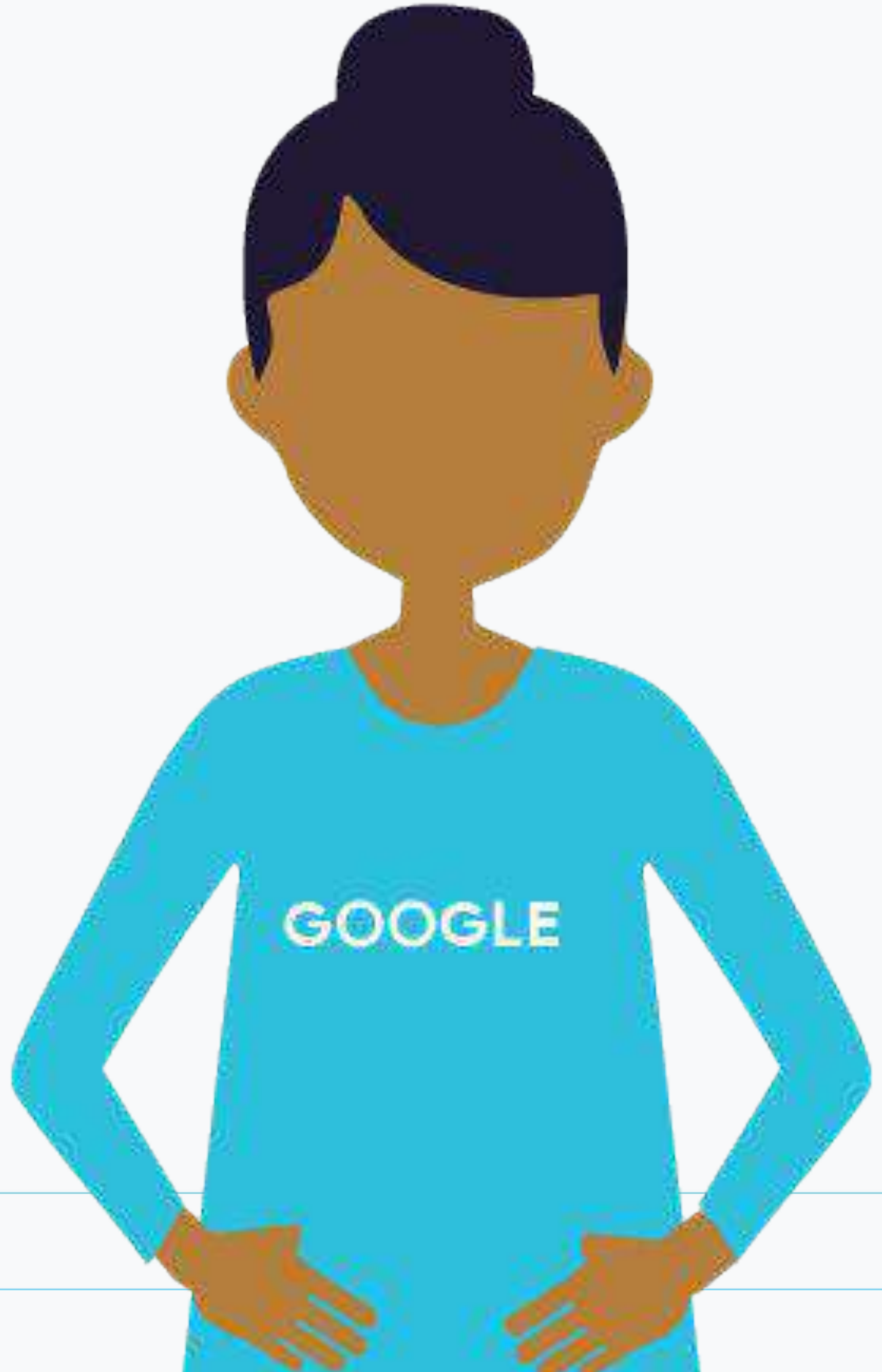
Client

/job:worker/task:0/

cpu:0

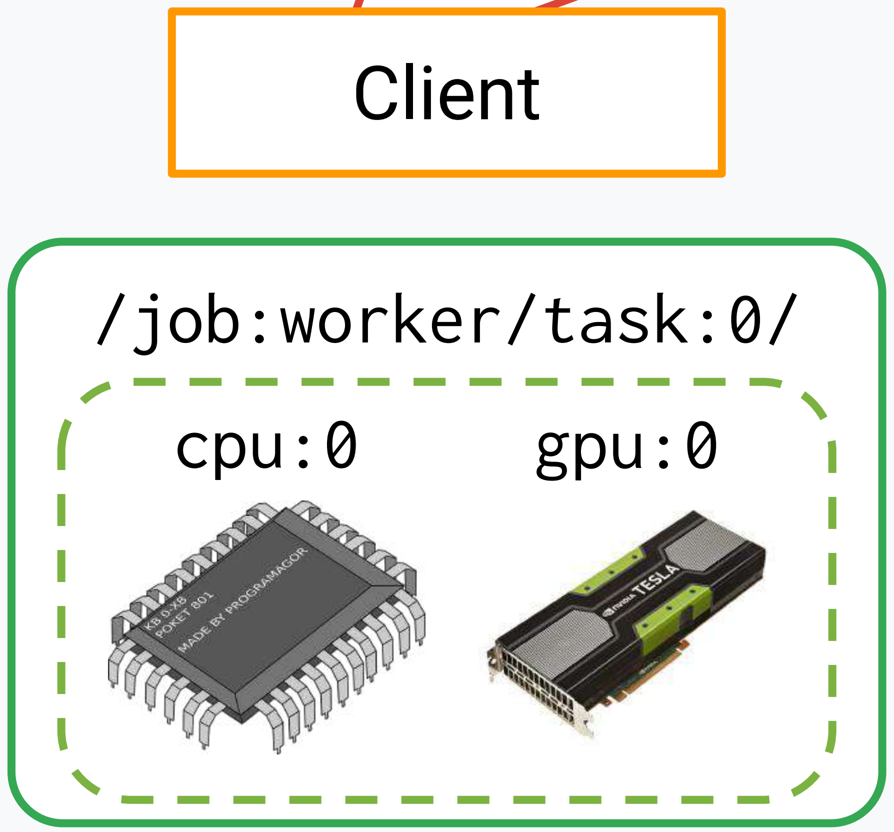


gpu:0




# TensorFlow inserts necessary data transfers

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



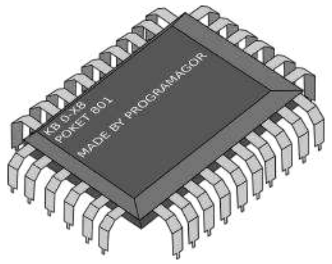
# TensorFlow inserts necessary data transfers

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

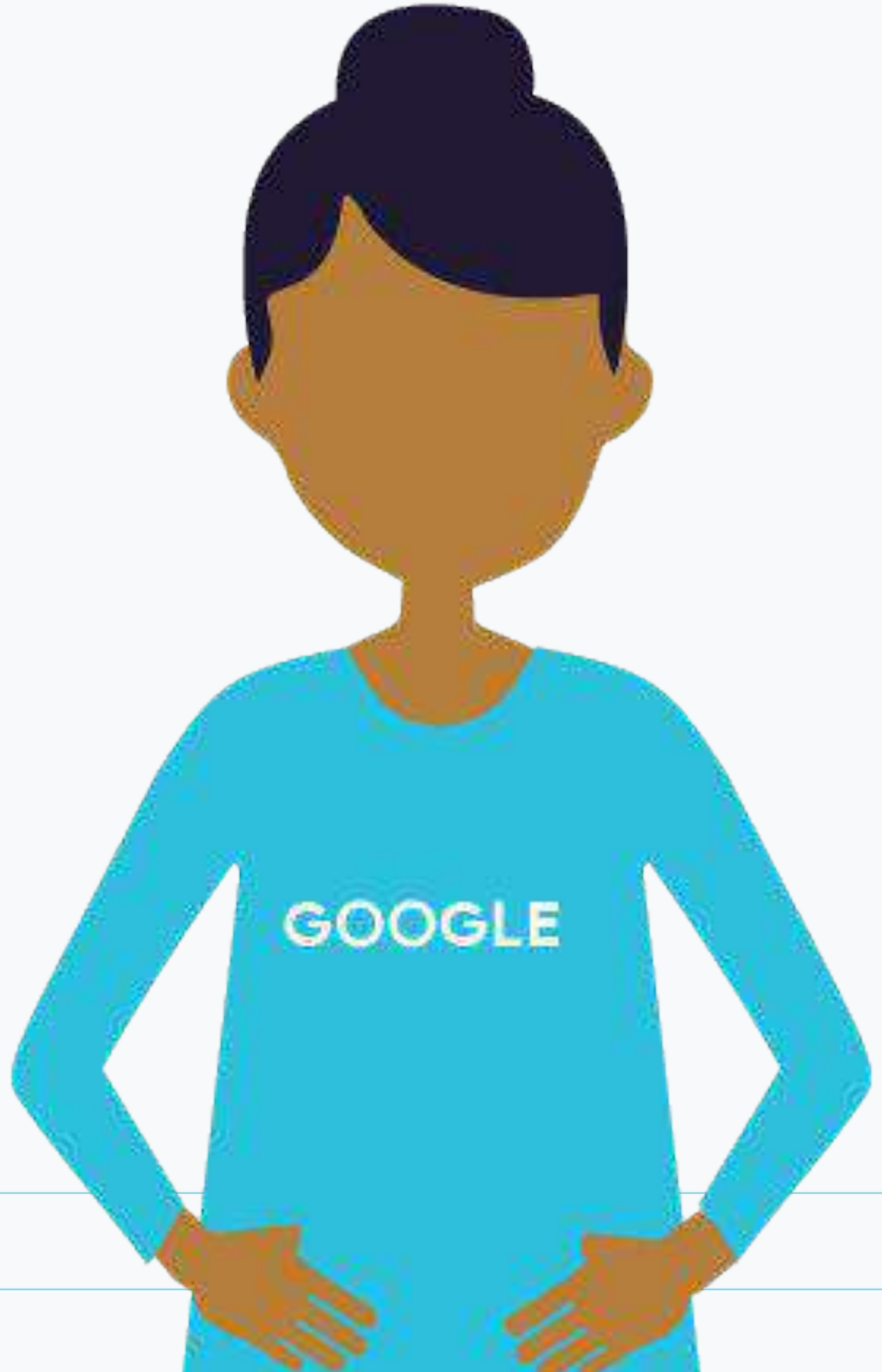
Client

/job:worker/task:0/

cpu:0



gpu:0



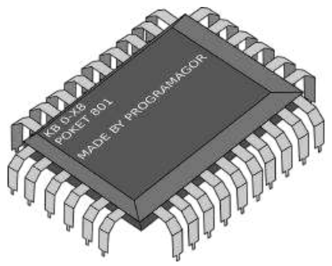

# What if you have a second machine?

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

/job:worker/task:0/

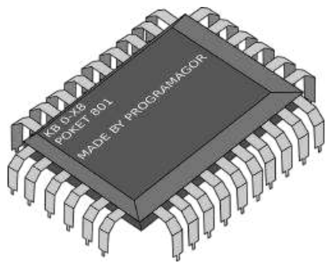
cpu:0



gpu:0

/job:ps/task:0/

cpu:0



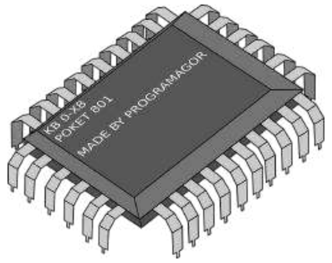
# What if you have a second machine?

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

/job:worker/task:0/

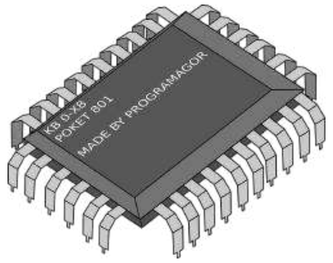
cpu:0

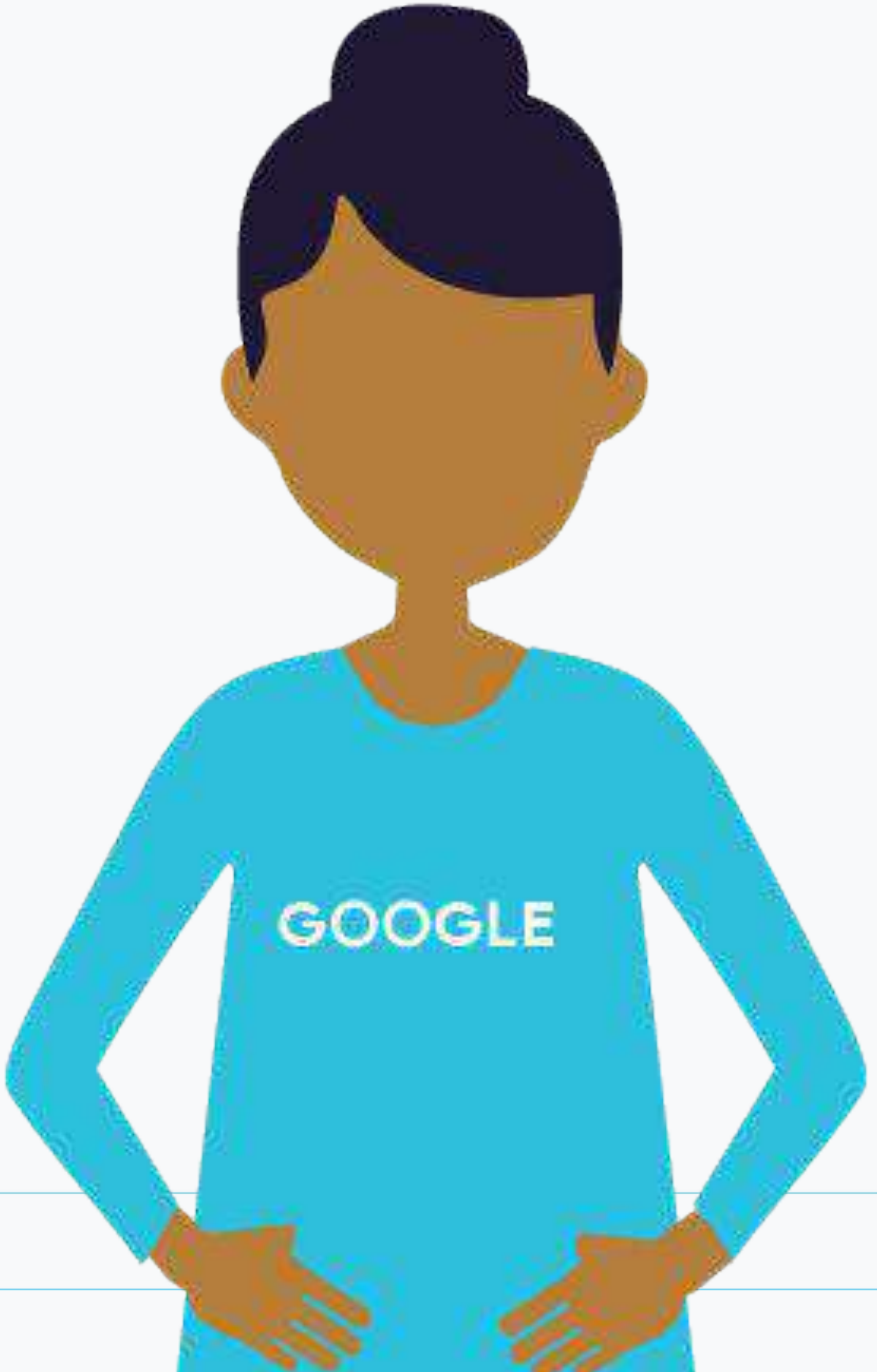


gpu:0

/job:ps/task:0/

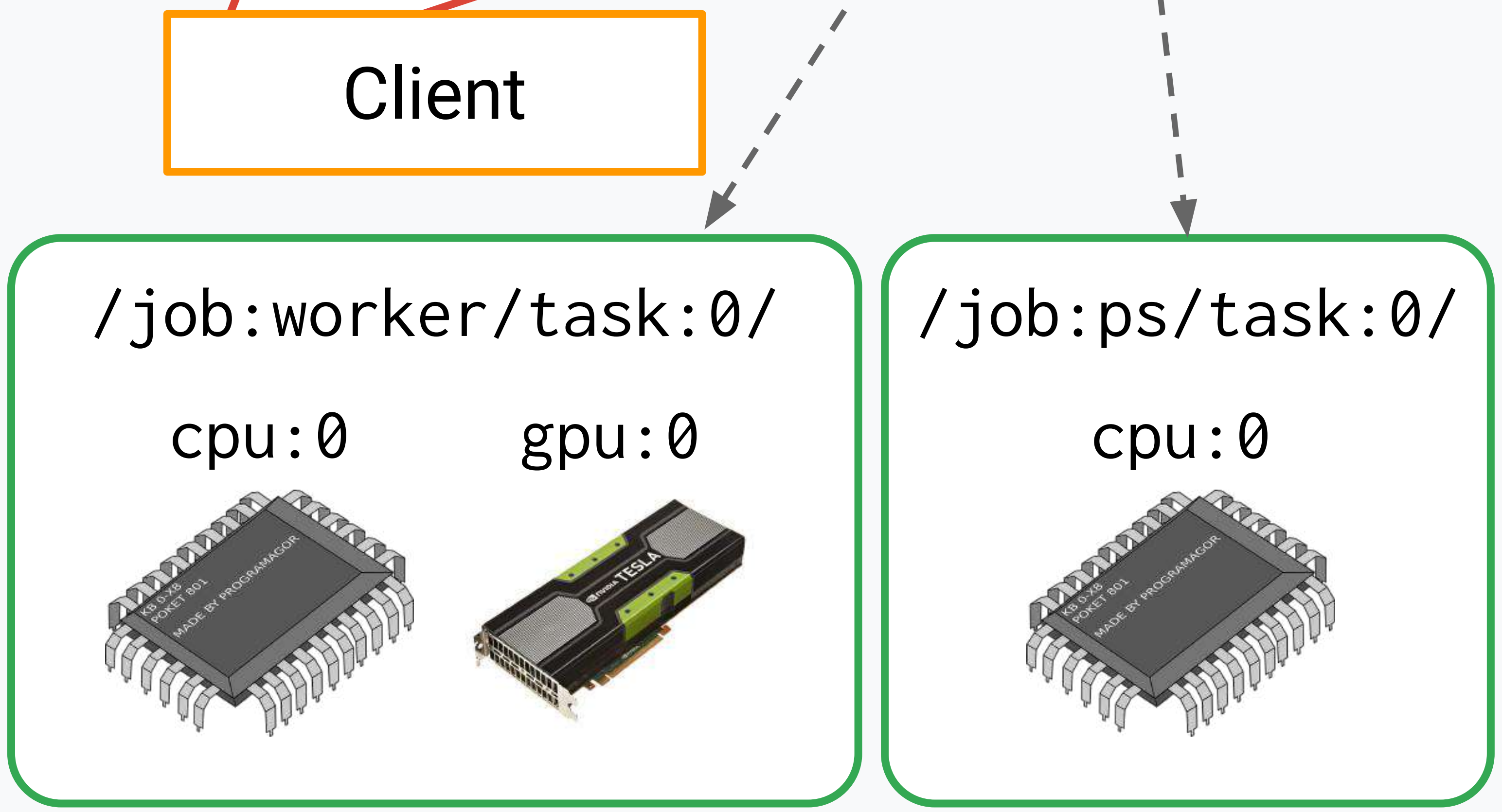
cpu:0





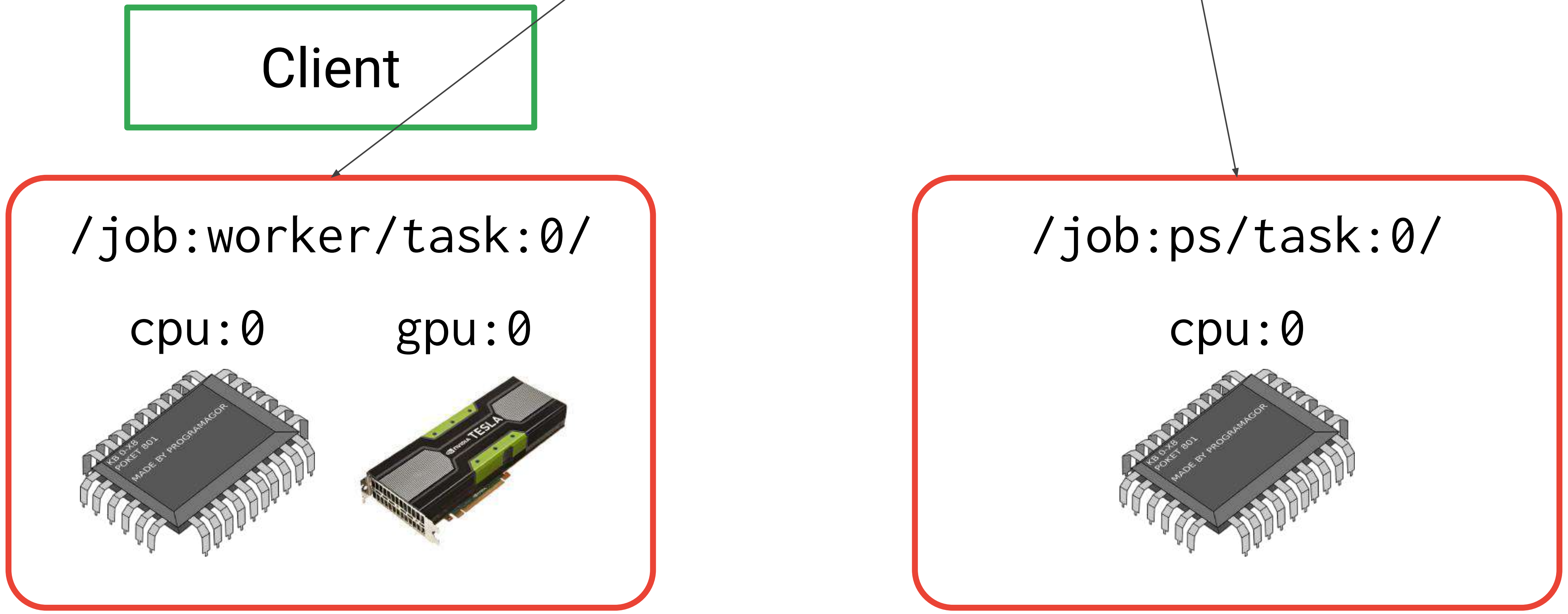

# Assign different tasks to different machines

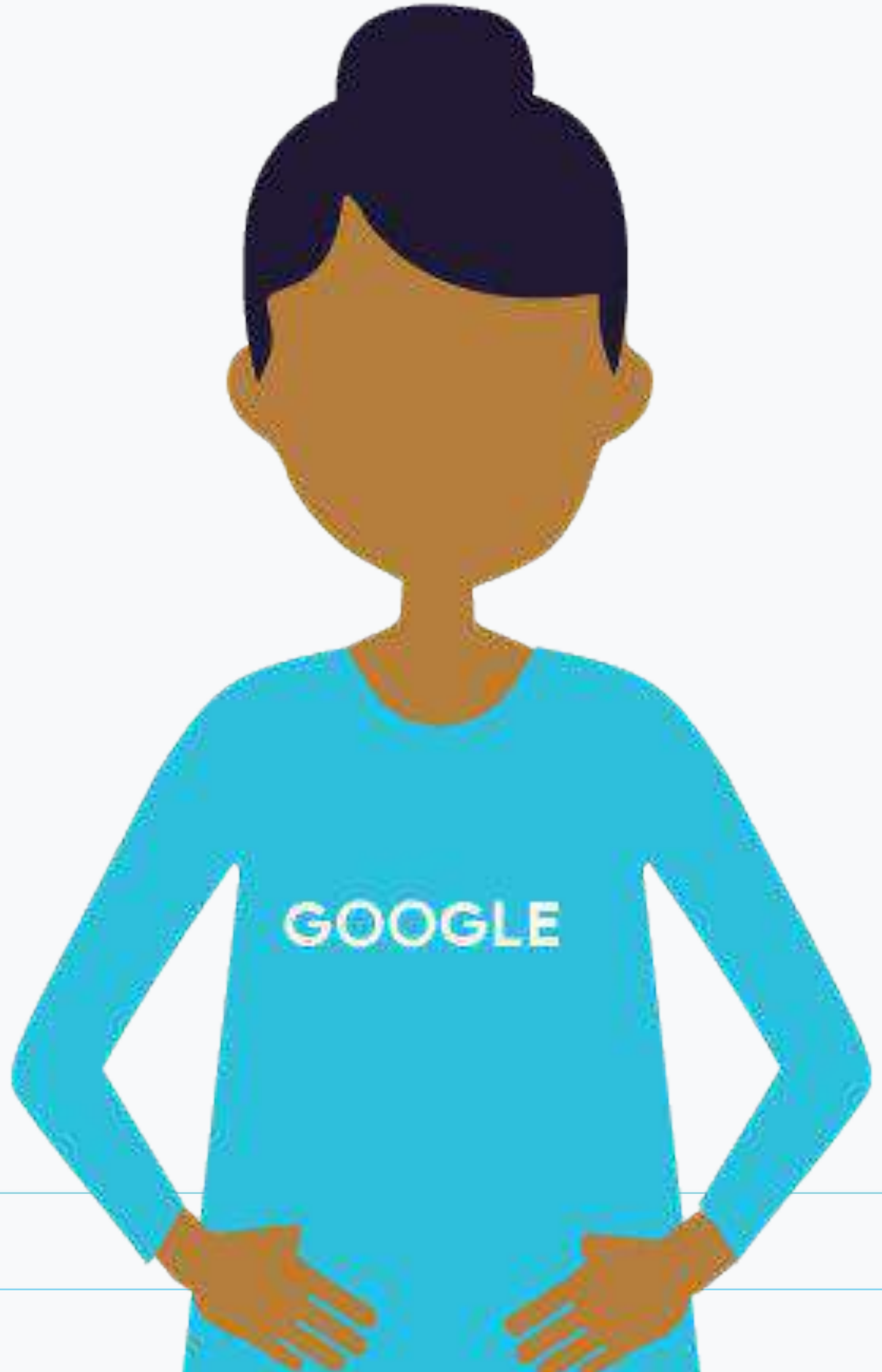
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



# Assign different tasks to different machines

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

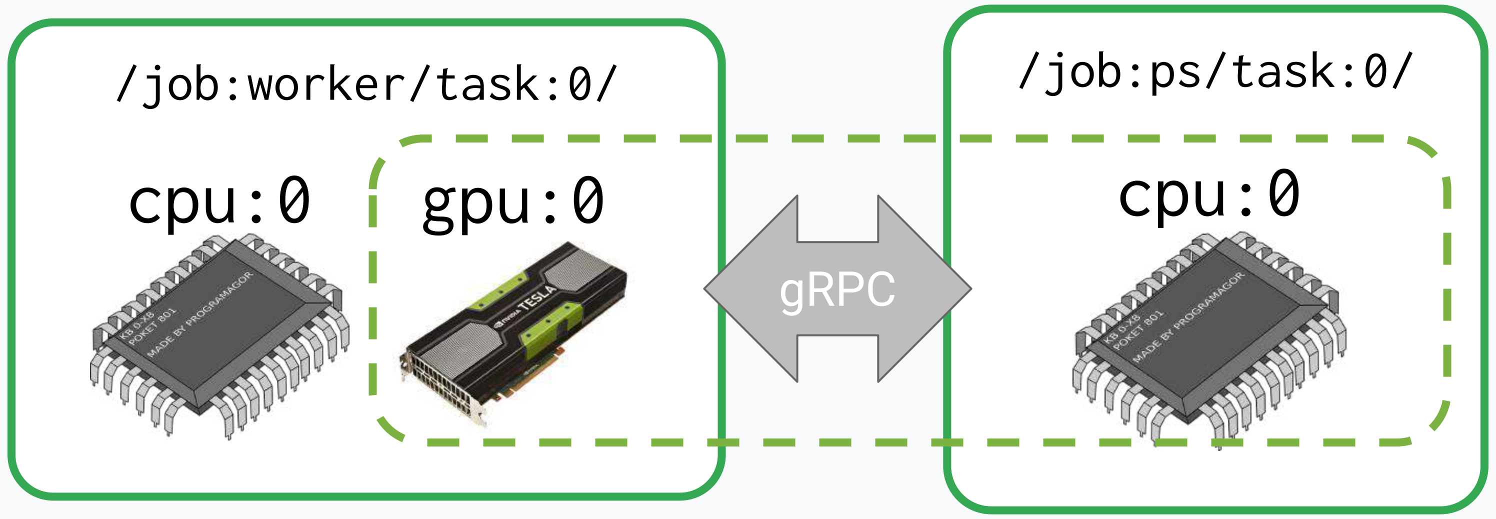





# Now, the graph is split between two processes ...

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

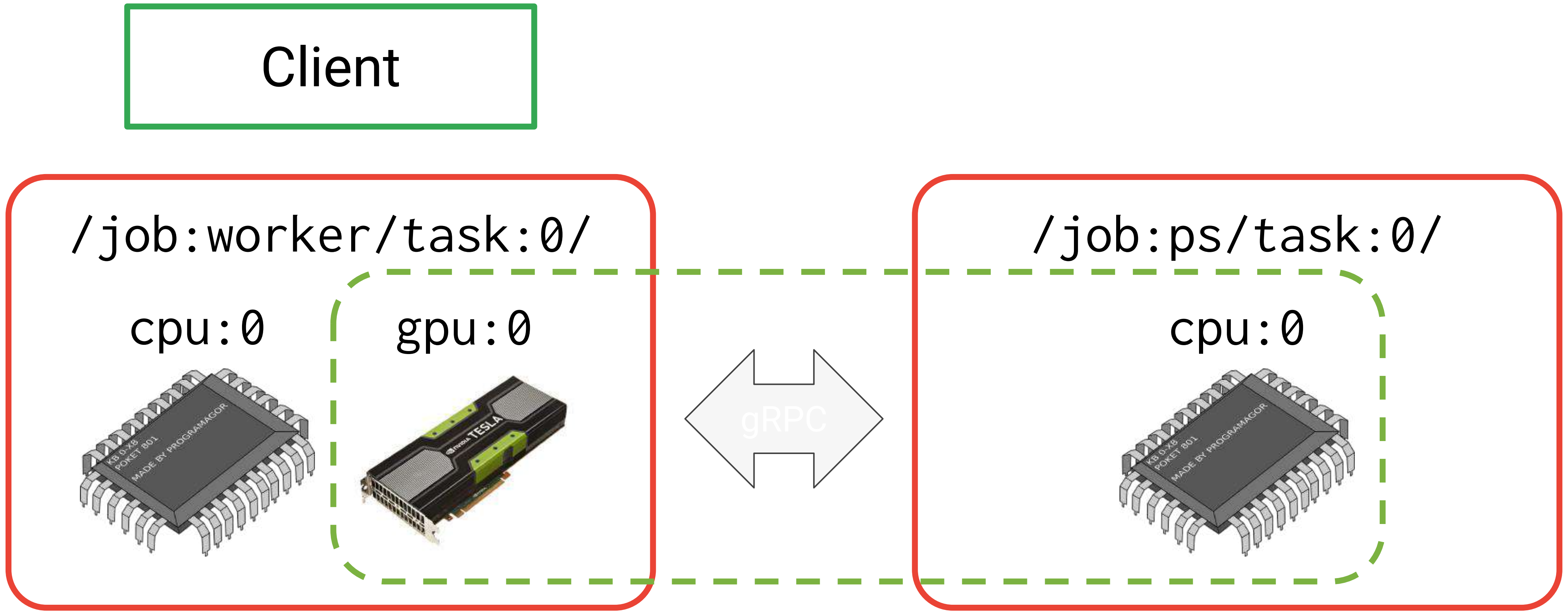



Title Safe >

< Action Safe

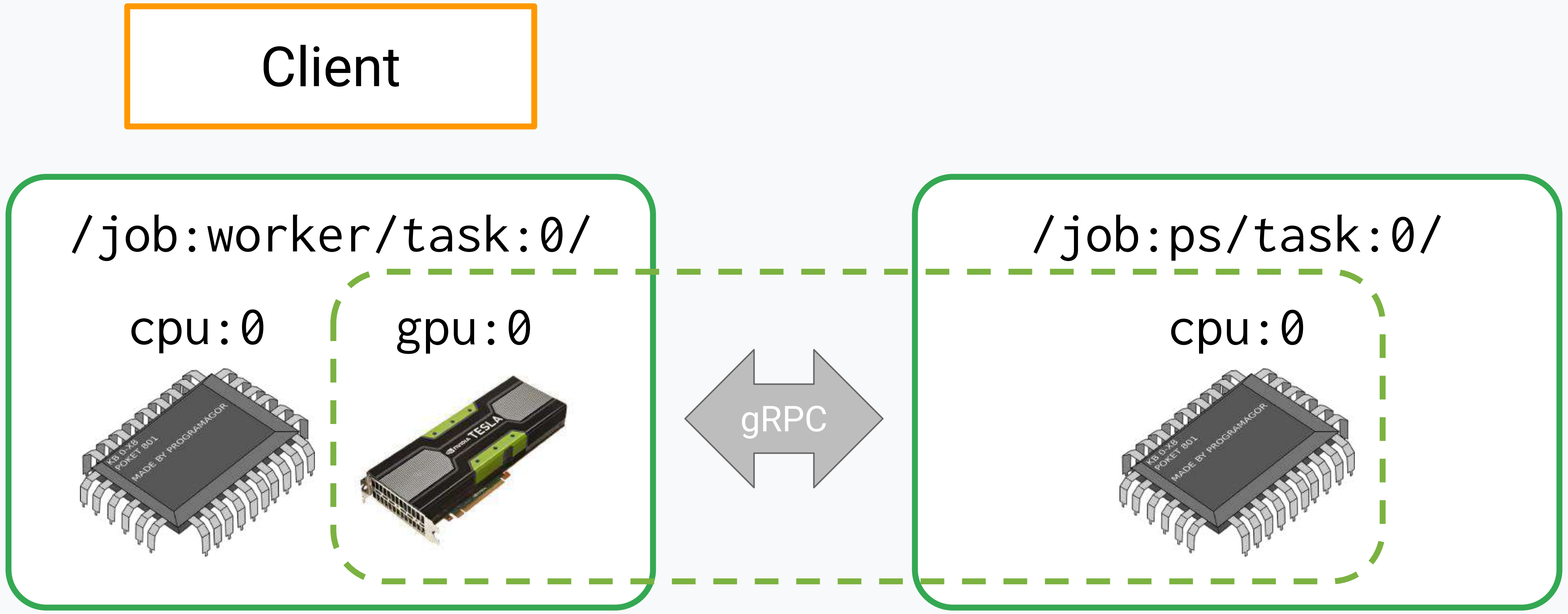
# Now, the graph is split between two processes ...

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



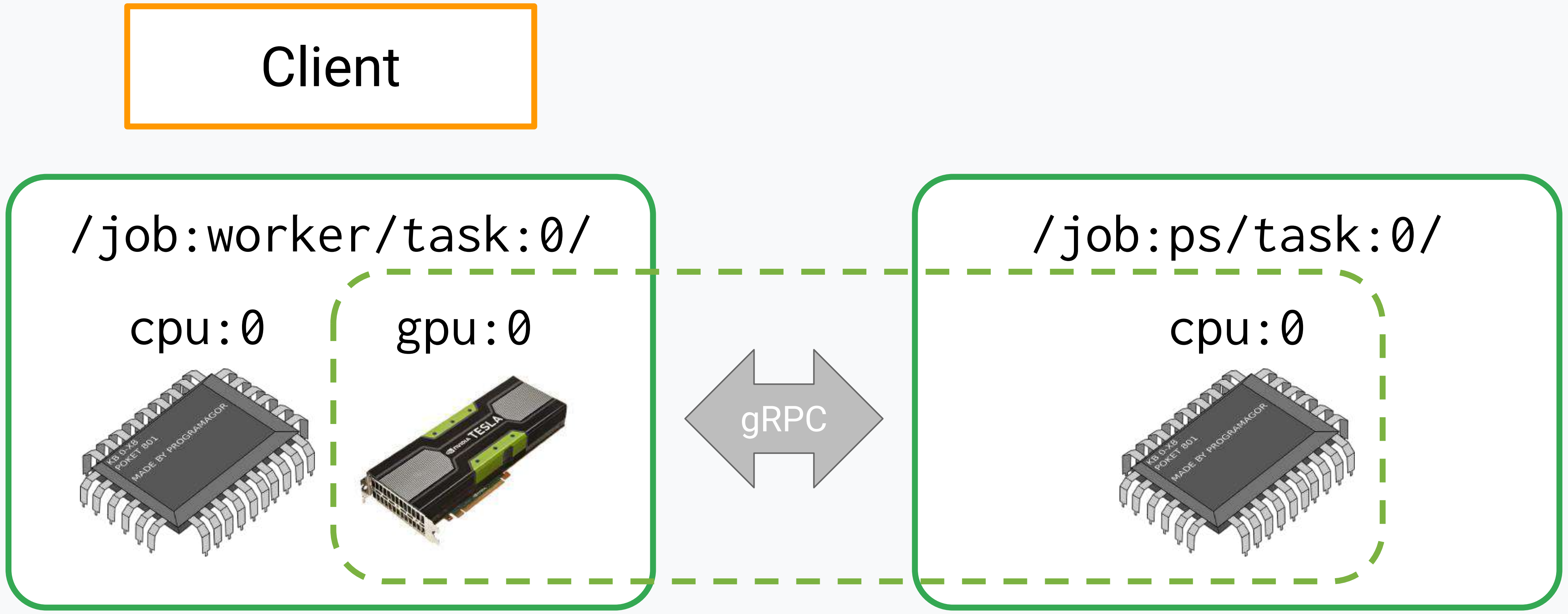
# ... we just get need to get the device placements right

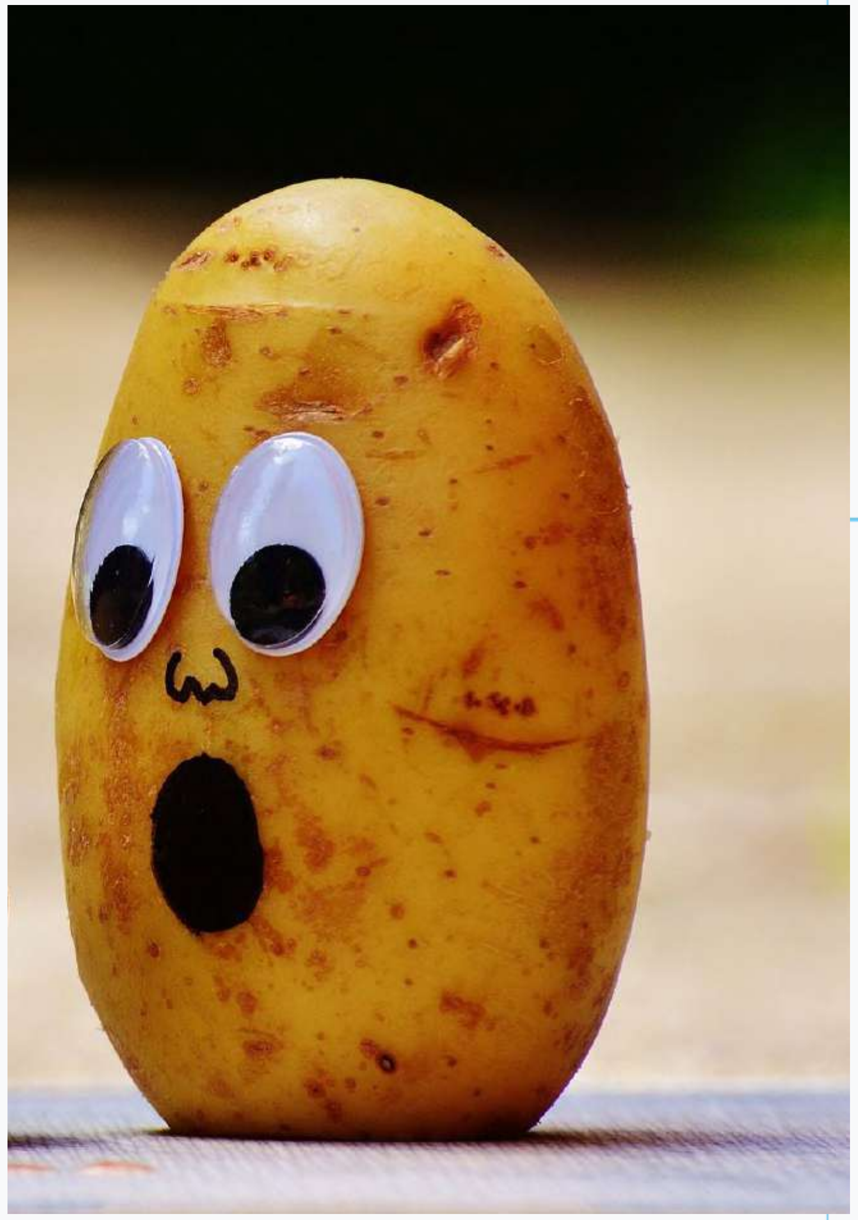
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



# … we just get need to get the device placements right

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Simplifying device placement [optional]**

Format: Screencast

Presenter: Laurence Moroney

Video Name:
T-PSML-O_4_l15_simplifying_device_placement_[optional]

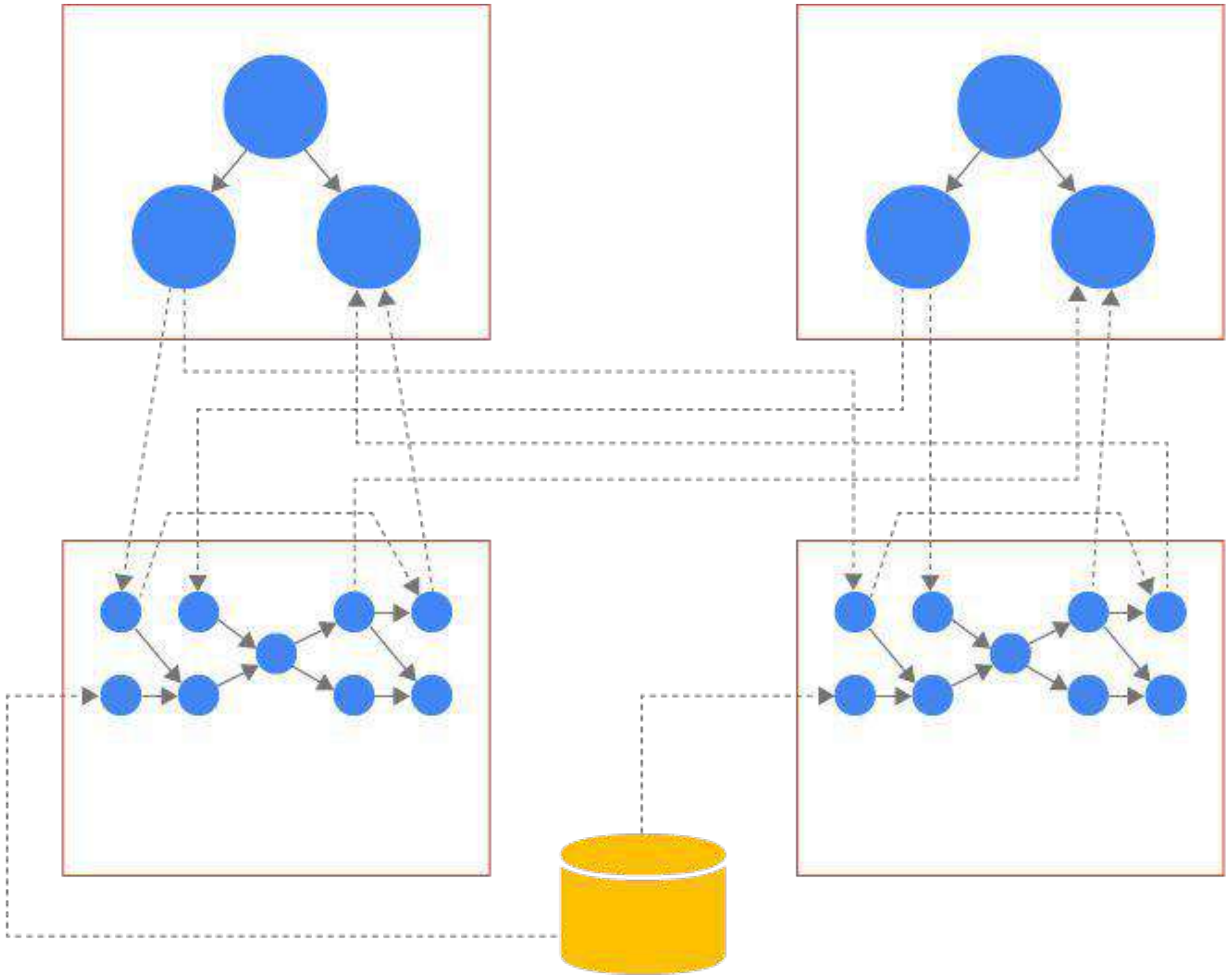
# One way to simplify this is to think in terms of parameter servers and worker replicas

PS tasks

- Variables
- Update ops

Worker tasks

- Pre-processing
- Loss calculation
- Backpropagation



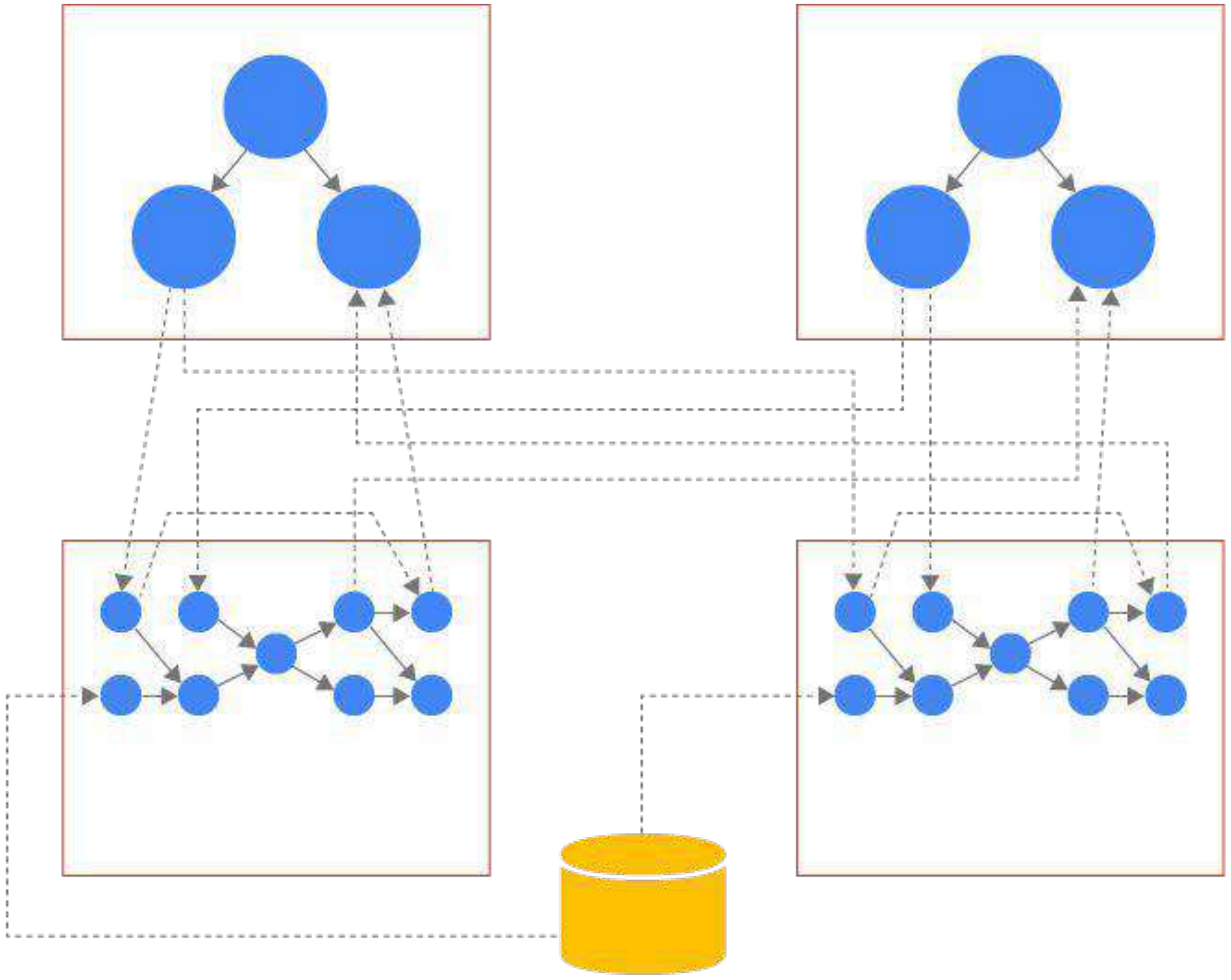
# In TensorFlow, it's the same program for both

PS tasks

- Variables
- Update ops

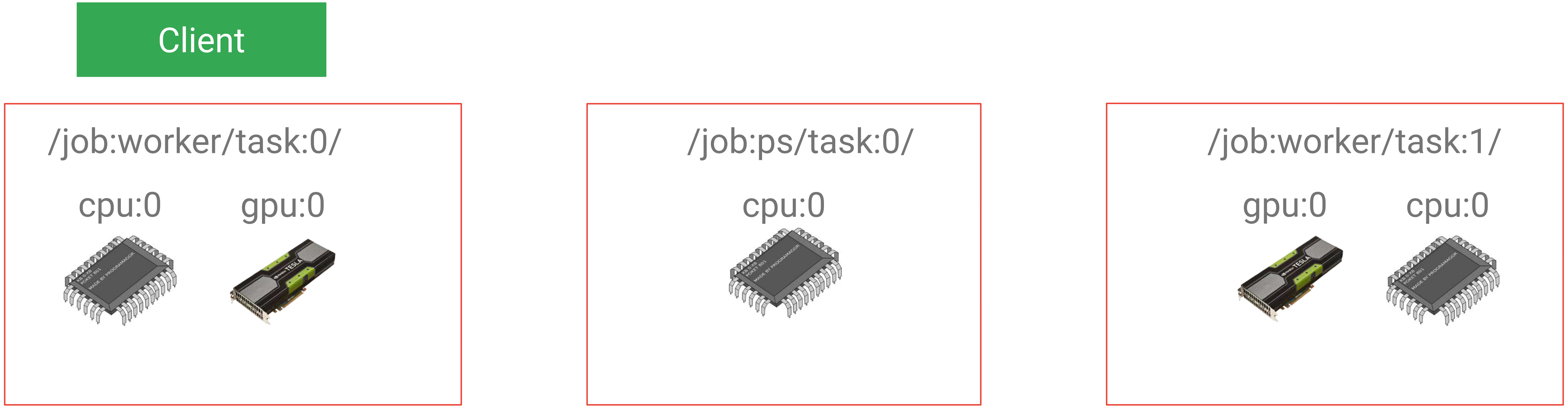
Worker tasks

- Pre-processing
- Loss calculation
- Backpropagation



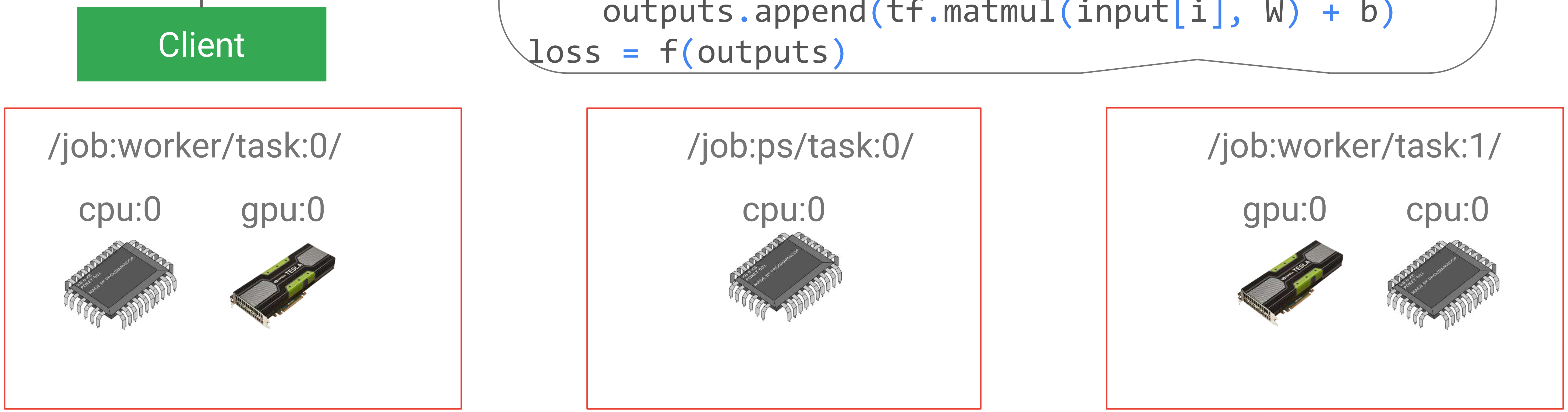
# Use in-graph replication: if a single step can be done on a single machine

**Each worker works on different subsets of the data**



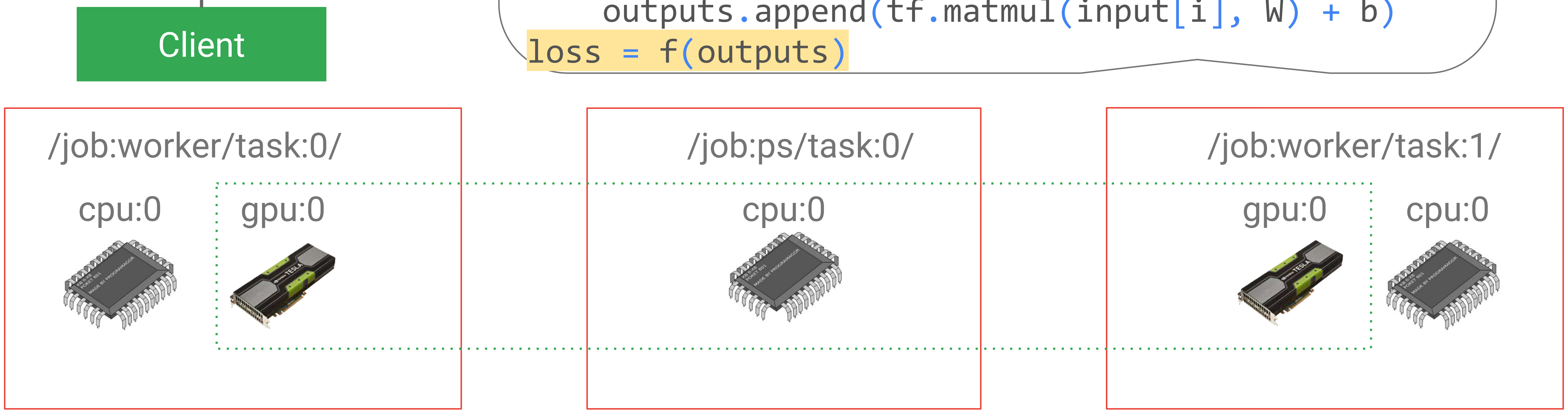
# With in-graph replication, data is broken into equal-sized chunks and each worker works on a chunk

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0, num_workers, input)
outputs = []
for i in range(num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" %
i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(outputs)
```



# Replicas are run in parallel, and loss is averaged across workers

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0, num_workers, input)
outputs = []
for i in range(num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" %
i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(outputs)
```



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Between-graph replication [optional]**

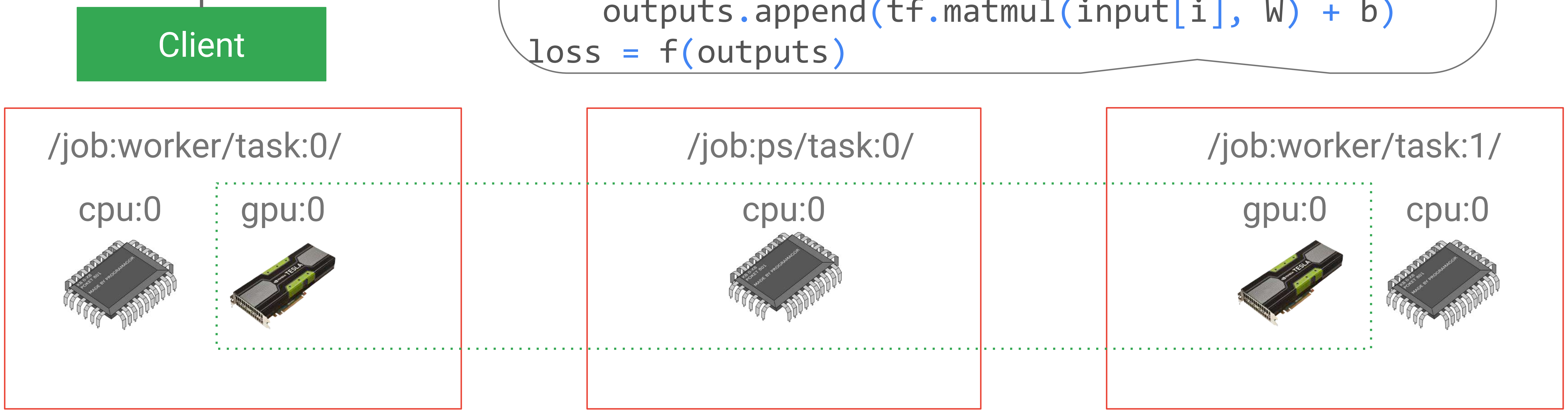
Format: Screencast

Presenter: Laurence Moroney

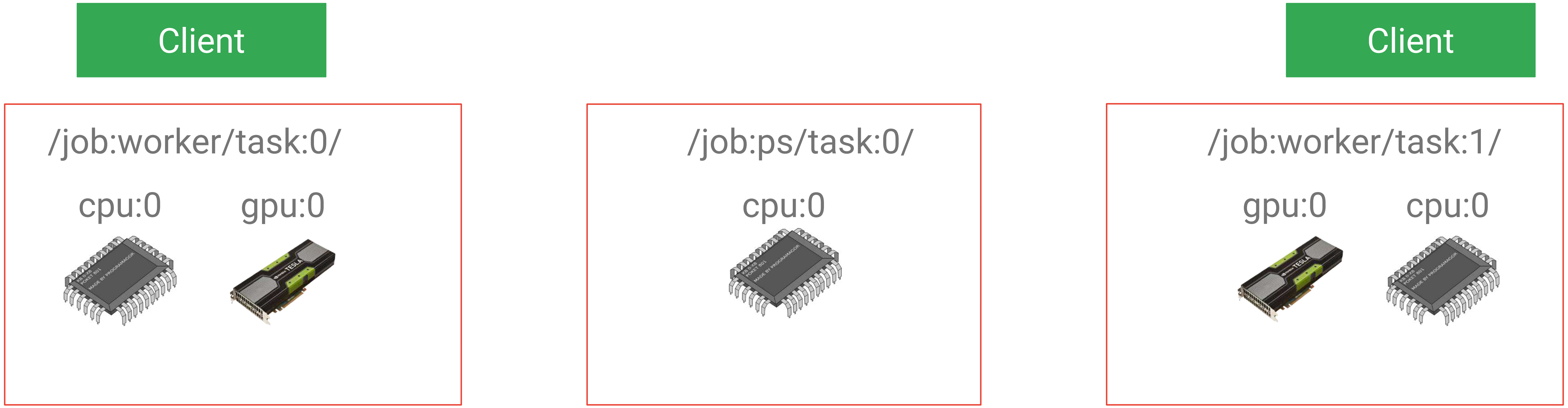
Video Name: T-PSML-O_4_l16_between-graph_replication_[optional]

# In-graph replication doesn’t work for very large models

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0, num_workers, input)
outputs = []
for i in range(num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" %
i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(outputs)
```



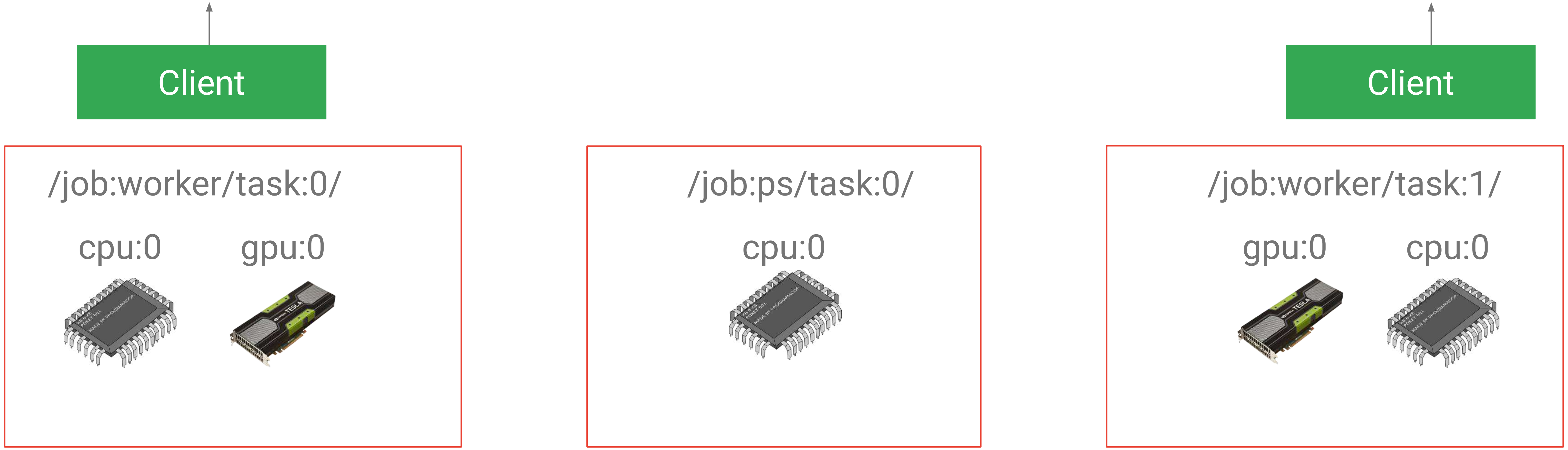
In between-graph replication, there are multiple client programs; this is the recommended approach in Cloud MLE



# Both clients build the same graph

```python
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

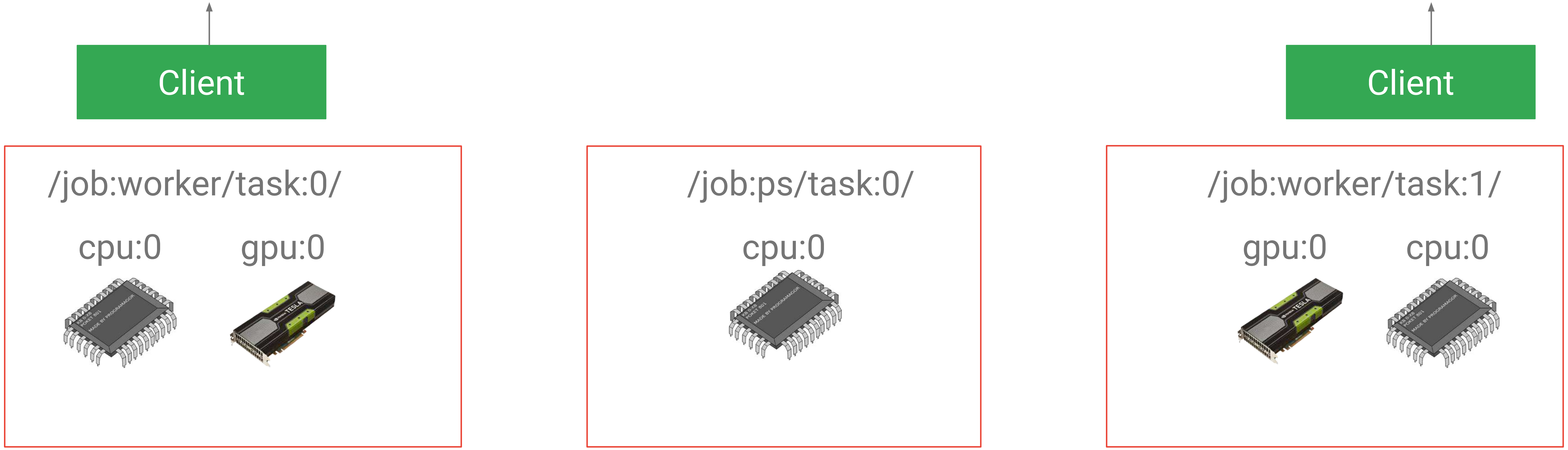
```python
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



# Local devices get a replica from “their” client

```python
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

```python
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



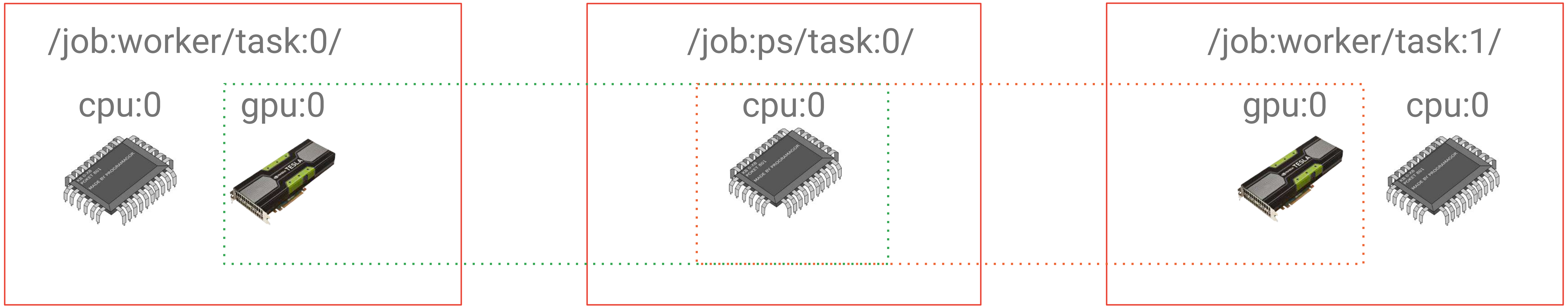
# Machines without a client share variables, leading to shared storage

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

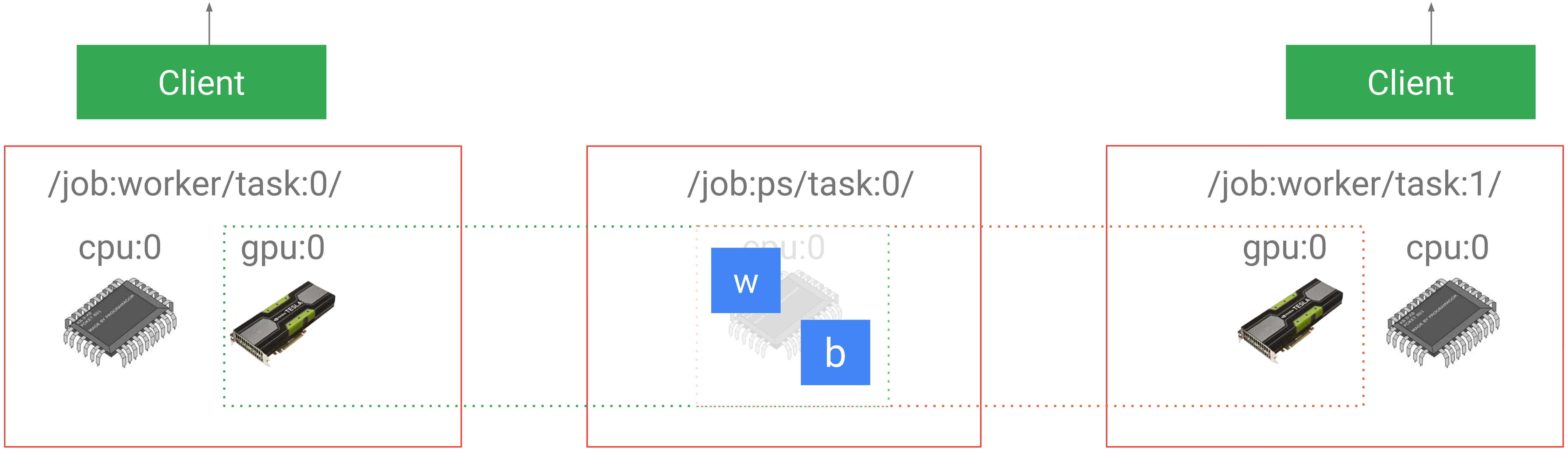
Client



# Here, weights and biases are shared, but loss is not

```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

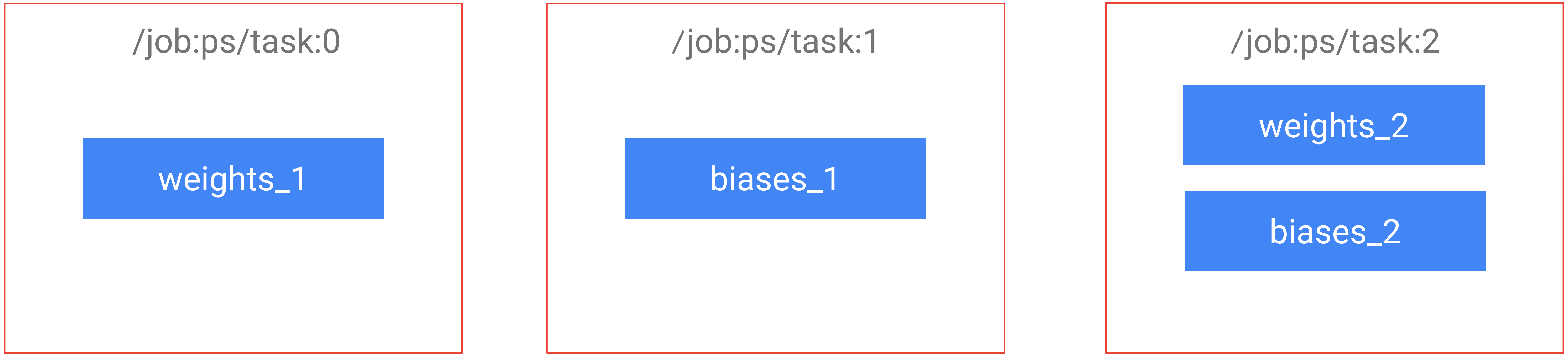
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



# Defining variable placement with strings is brittle

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.get_variable("weights_1", [784, 100])
with tf.device("/job:ps/task:1"):
  biases_1 = tf.get_variable("biases_1", [100])
with tf.device("/job:ps/task:2"):
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

**Hardcoding strings in your program is a bad idea**



Courses 7 - Production ML Systems

Module 4: Designing High-Performance ML Systems

Lesson Title: **Device Placement Strategies [optional]**

Format: Screencast

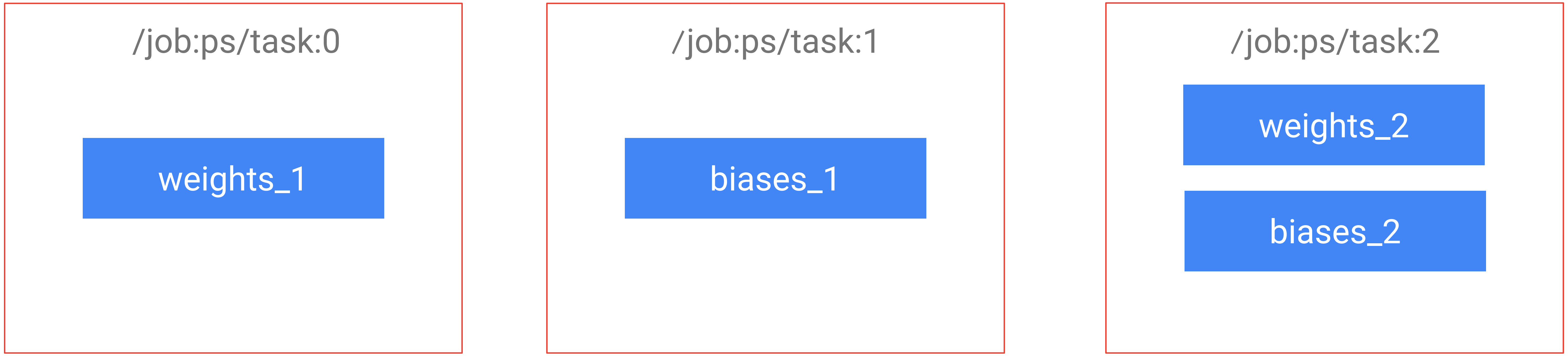
Presenter: Laurence Moroney

Video Name: T-PSML-O_4_l17_device_placement_strategies_[optional]

# Defining variable placement with strings is brittle

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.get_variable("weights_1", [784, 100])
with tf.device("/job:ps/task:1"):
  biases_1 = tf.get_variable("biases_1", [100])
with tf.device("/job:ps/task:2"):
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

**Hardcoding strings in your program is a bad idea**



# Use device placement functions instead

```
with
tf.device(tf.train.replica_device_setter(ps_tasks=3)):

  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

**Variables are assigned in round-robin fashion, so this is between-graph replication**

/job:ps/task:0

- weights_1
- biases_2

/job:ps/task:1

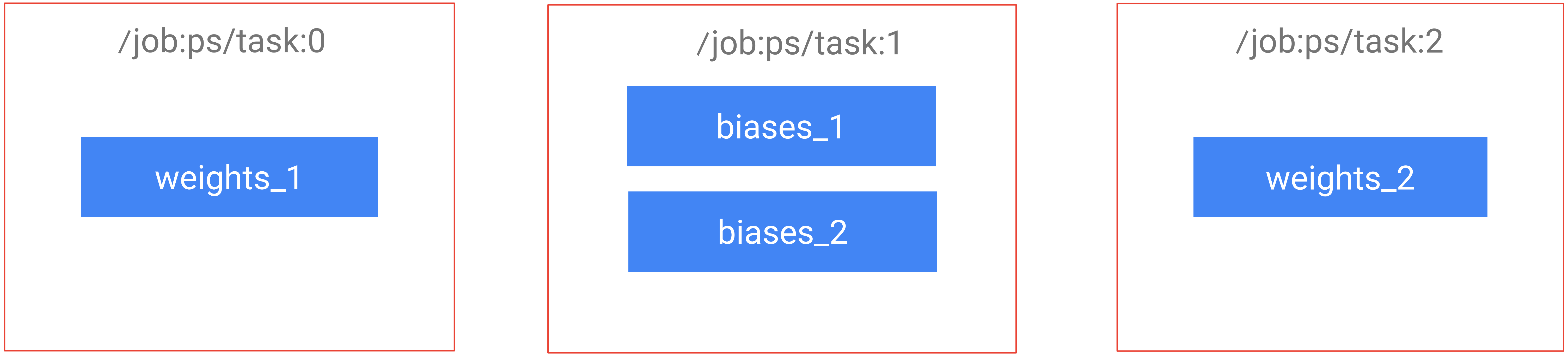
- biases_1

/job:ps/task:2

- weights_2

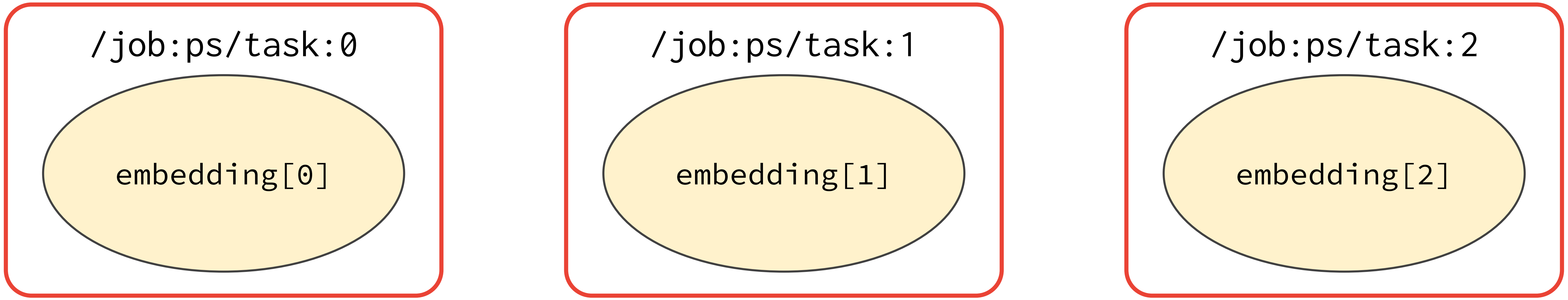
# A load balancing strategy can improve performance

```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(...)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):
  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

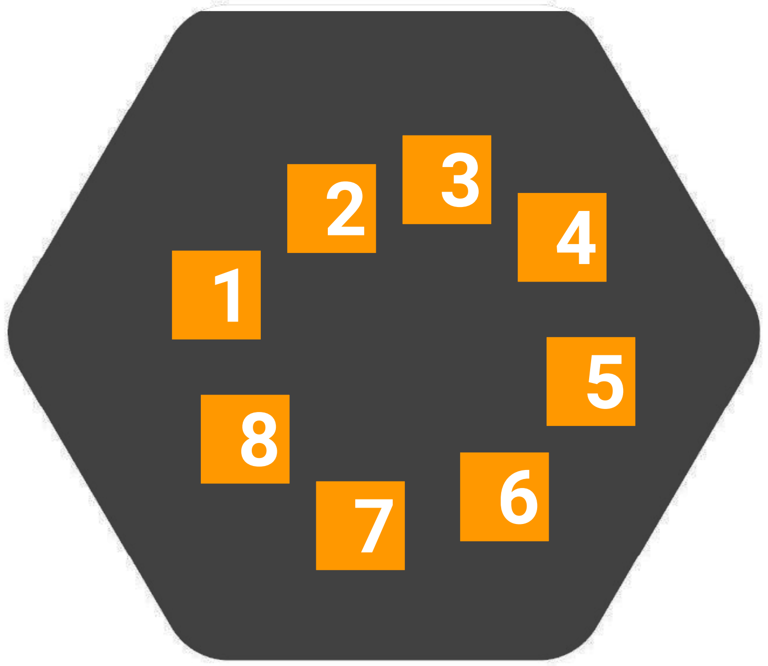


# You can partition very large variables as needed

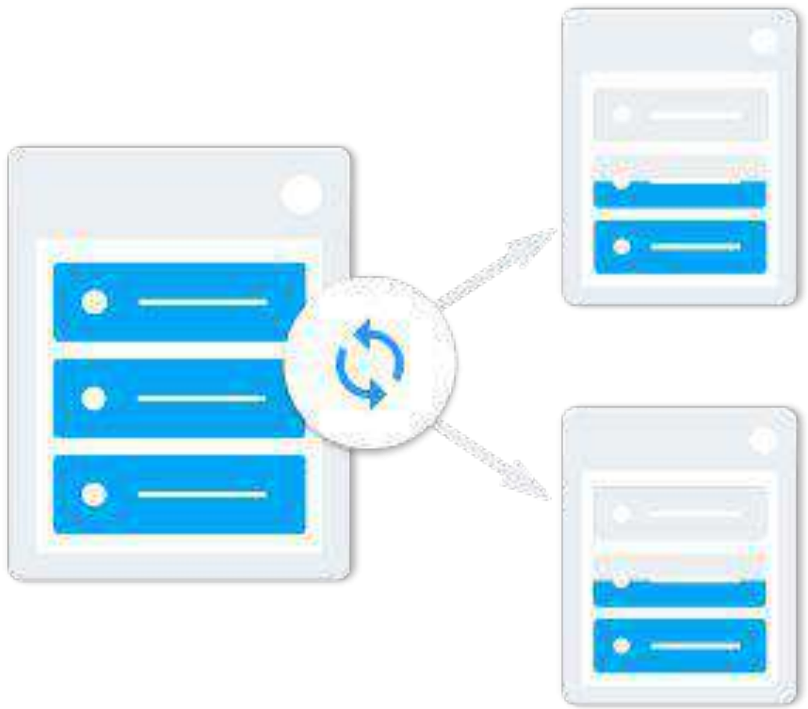
```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(...)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):

  embedding = tf.get_variable(
      "embedding", [1000000000, 20],
      partitioner=tf.fixed_size_partitioner(3))
```



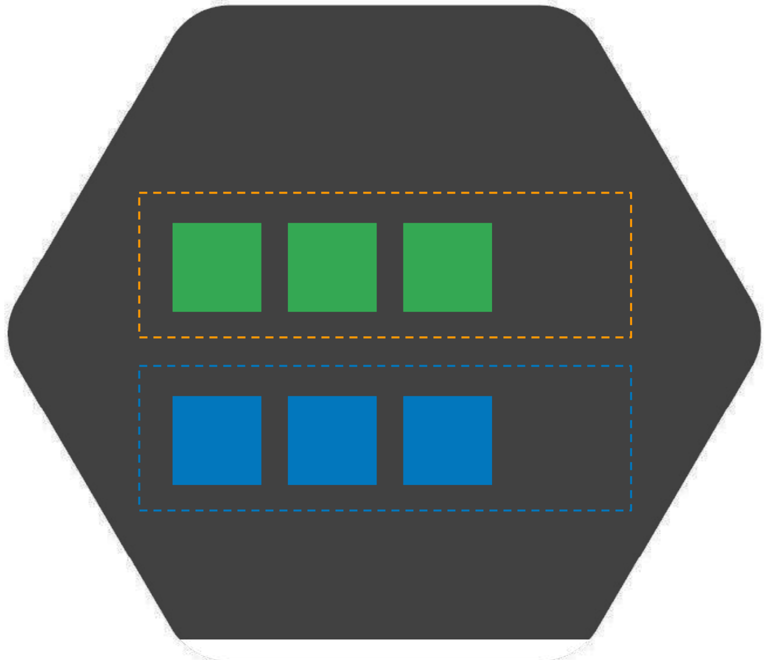
# `replica_device_setter` provides a simple heuristic for between-graph partitioning




Round-robin variable placement by default



Add load balancing to improve weight placement across machines



Partition if your variables are too large to fit on a single machine

Courses 7 - Production ML Systems

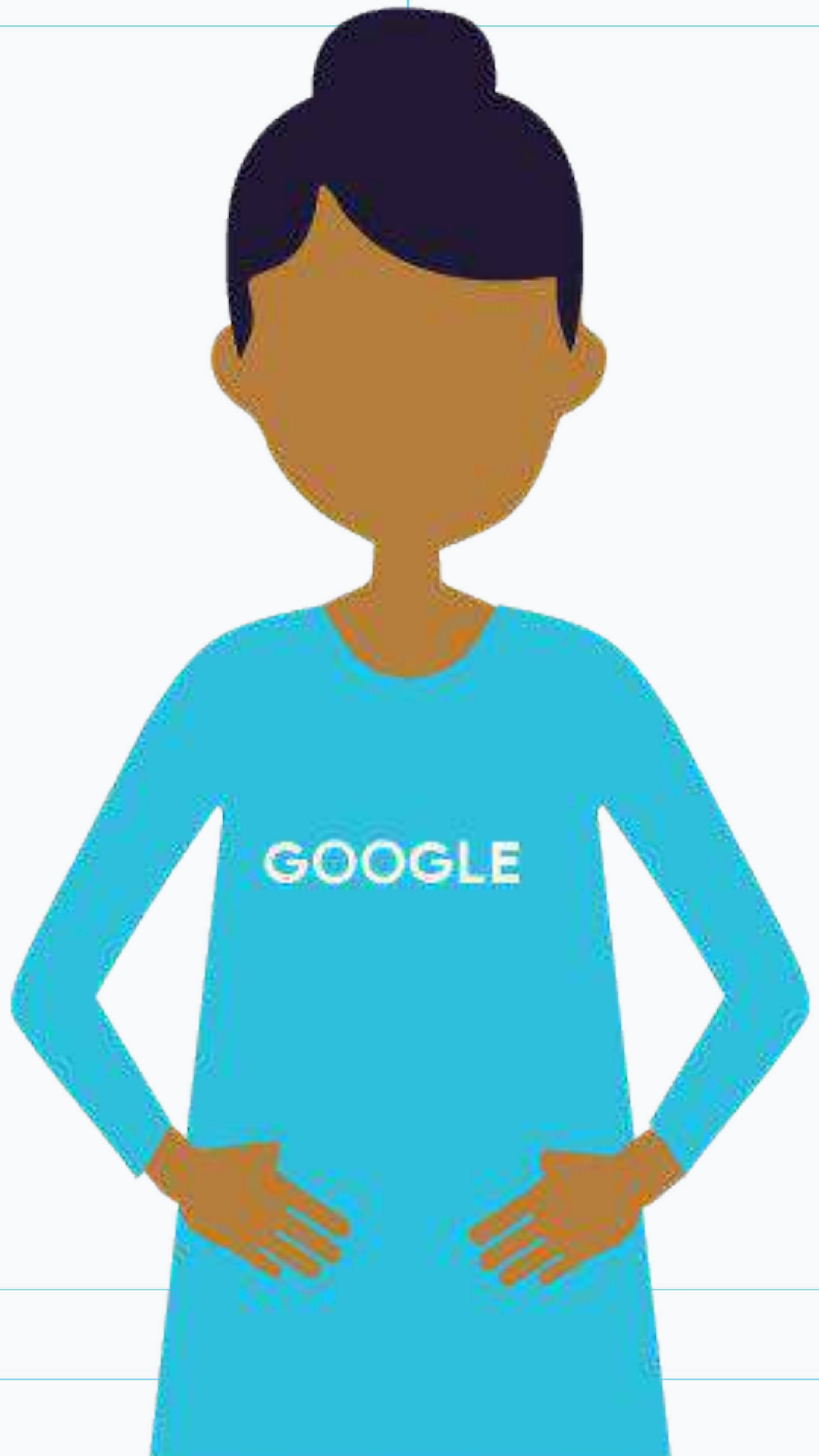
Module 4: Designing High-Performance ML Systems

Lesson Title: **Sessions and Servers [optional]**

Format: Presenter

Presenter: Laurence Moroney

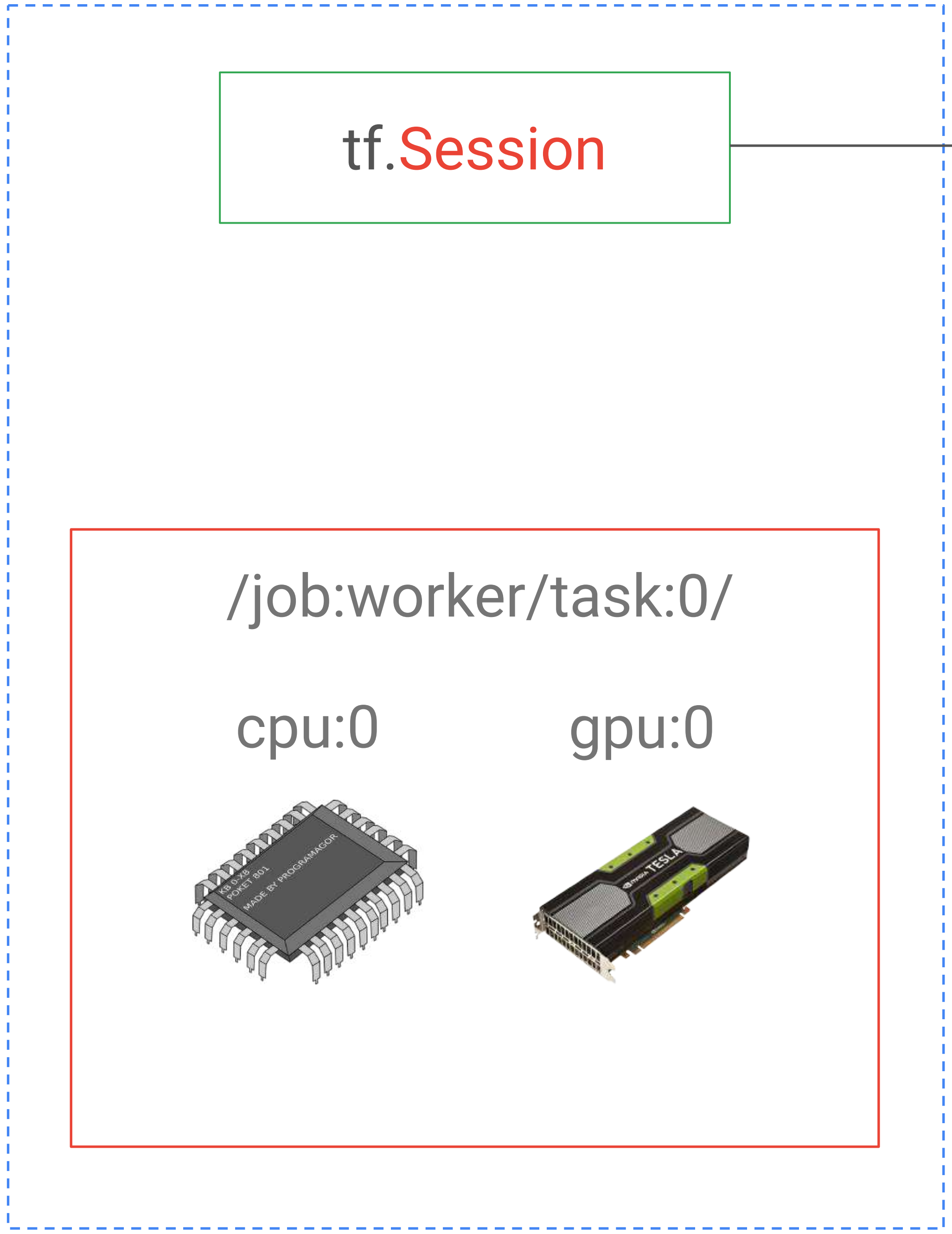
Video Name: T-PSML-O_4_l18_sessions_and_servers_[optional]




Title Safe >

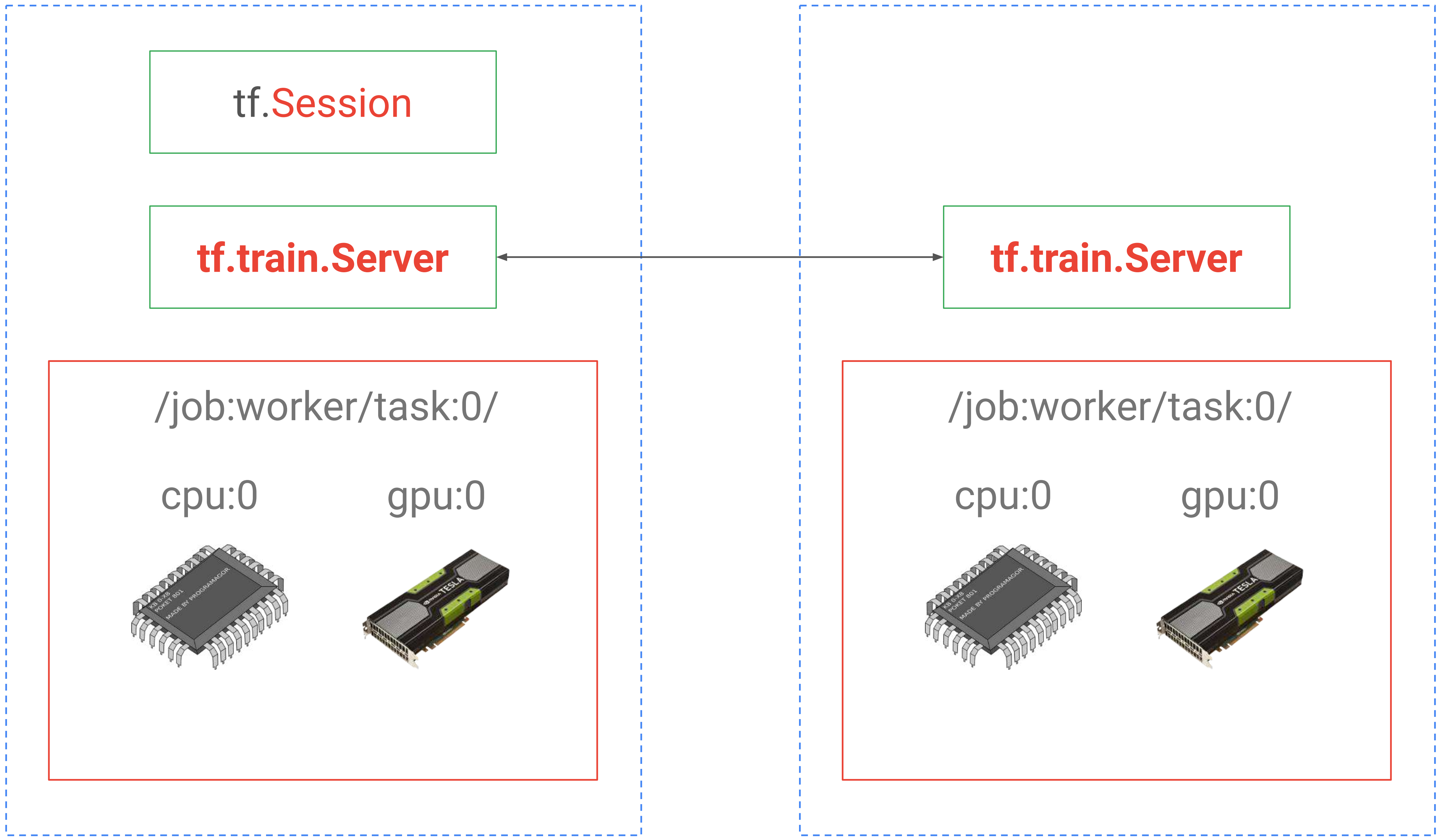
< Action Safe

# Distributed TensorFlow runs on a cluster of servers, but `tf.Session` knows only about local devices




```python
# Single-process code.
with tf.Session() as sess:
  sess.run(init_op)
  for _ in range(NUM_STEPS):
    sess.run(train_op)
```

# Create a TensorFlow server on each machine, and configure them to communicate over network



# The setup is defined by a cluster spec that is identical for each machine in the cluster

1. ClusterSpec: set of jobs (worker, ps)

2. For each job, specify a list of tasks

3. For each task, specify machine : network port to listen on

```
# Distributed code for a worker task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="worker", task_index=0)

with tf.Session(server.target) as sess:
  # …
```

# tf.train.Server implements a task in cluster

```
# Distributed code for a worker task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="worker", task_index=0)

with tf.Session(server.target) as sess:
  # …
```

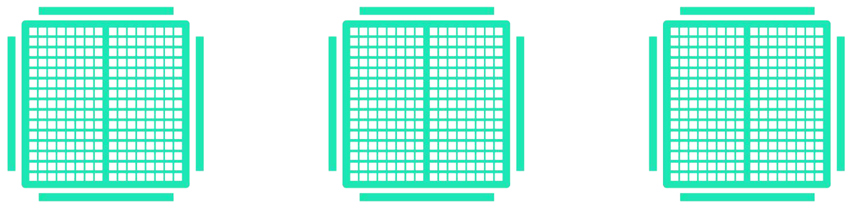
Which job, which task

Can run code on any device in cluster

# A PS task simply needs to respond to workers

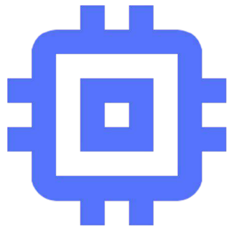
```
# Distributed code for a PS task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", ...],
                                "ps": ["192.168.1.1:2222", ...]})

server = tf.train.Server(cluster, job_name="ps", task_index=0)

# Wait for incoming connections forever.
server.join()
```

# Recap: Create a cluster spec for two types of jobs



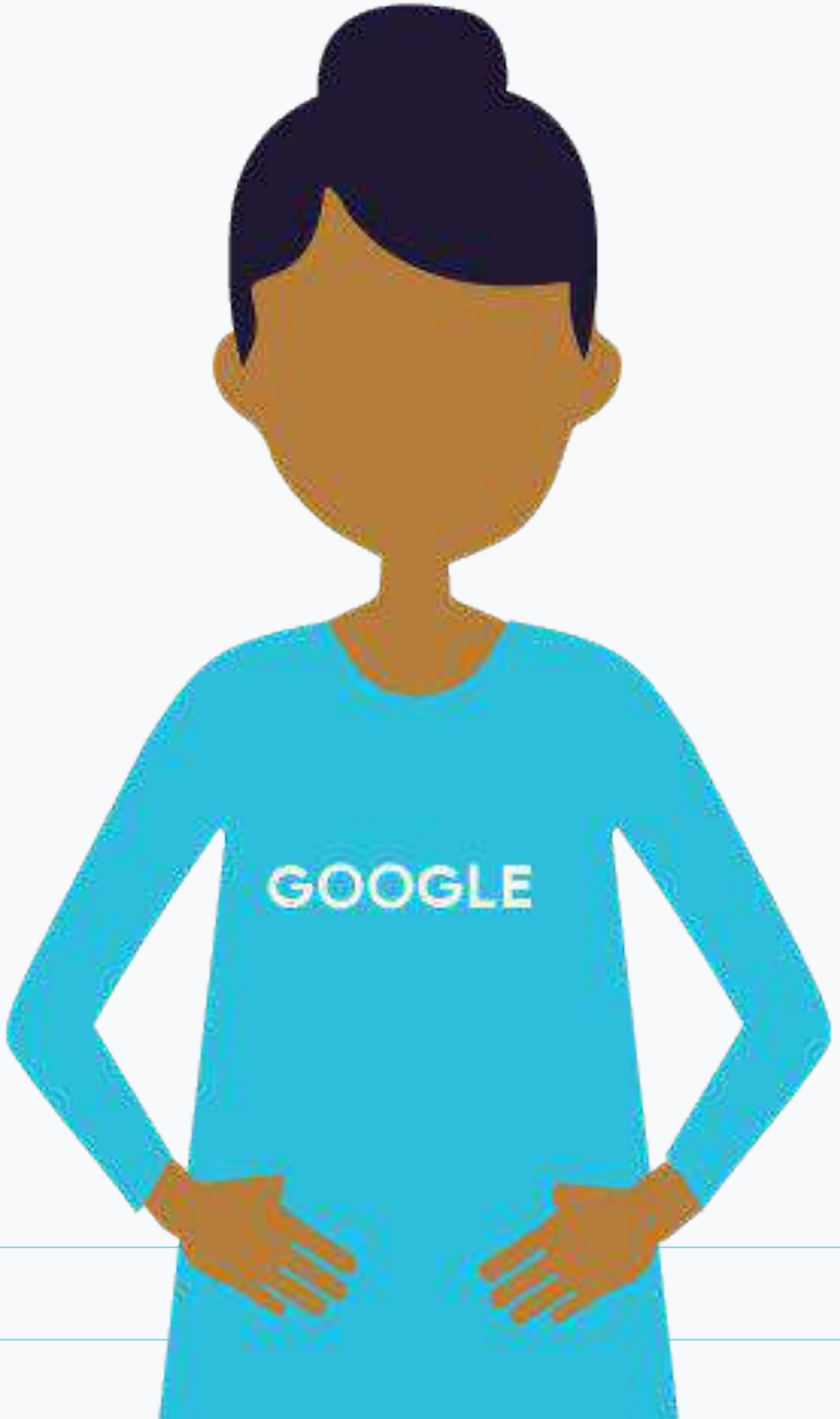
## Worker jobs

- Each worker has a `tf.train.Server`.
- Specify the server's own address as the Session target.
- Can run code on any device in the cluster.



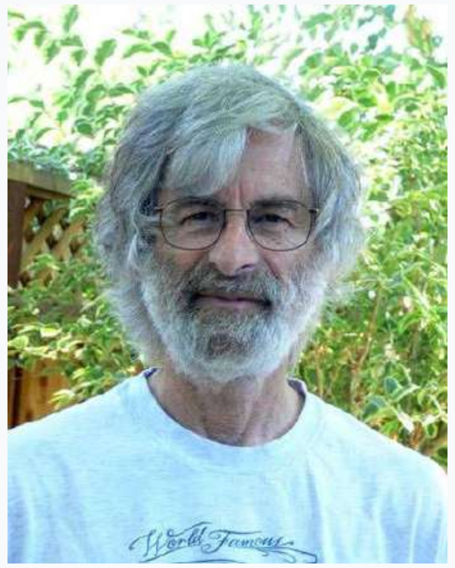
## Parameter Server (PS) jobs

- Simply call `server.join()`




*"A **distributed system** is a system where I can't get my work done because a computer has failed that I've never even heard of."*

*Leslie Lamport*



Source
https://upload.wikimedia.org/wikipedia/commons/5/50/Leslie_Lamport.jpg

Title Safe >

< Action Safe

Courses 7 - Production ML Systems

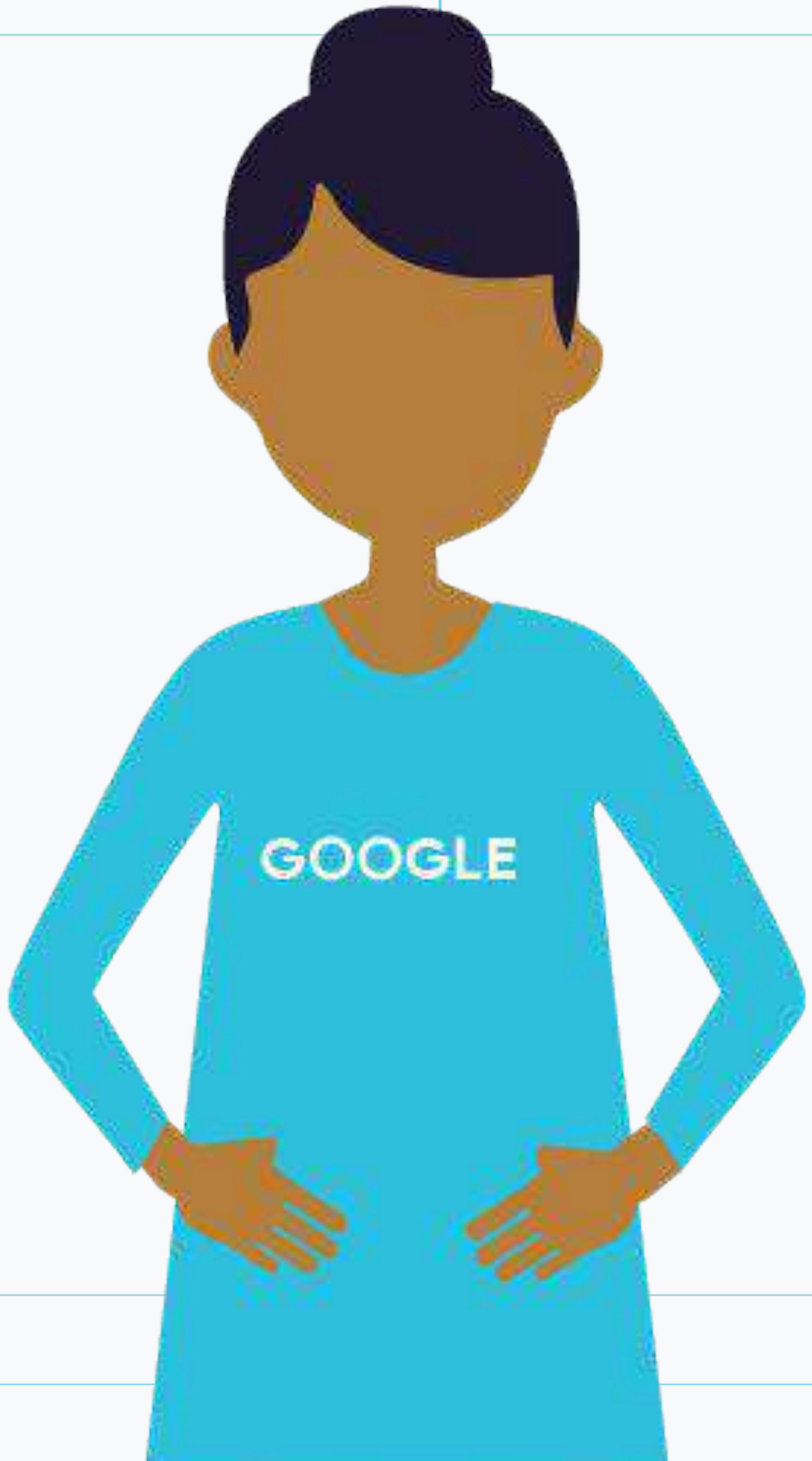
Module 4: Designing High-Performance ML Systems

Lesson Title: **Fault Tolerance [optional]**

Format: Presenter

Presenter: Laurence Moroney

Video Name: T-PSML-O_4_I19_fault_tolerance_[optional]




Title Safe >

< Action Safe

# For fault tolerance, save sharded checkpoints to Google Cloud Storage

```
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):
  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  # ...

saver = tf.train.Saver(sharded=True)

with tf.Session(server.target) as sess:
  while True:
    # ...
    if step % 1000 == 0:
      saver.save(sess, "gs://mybucket/chk_{}".format(step))
```

Each PS task writes “its” variables; together the shards form a checkpoint

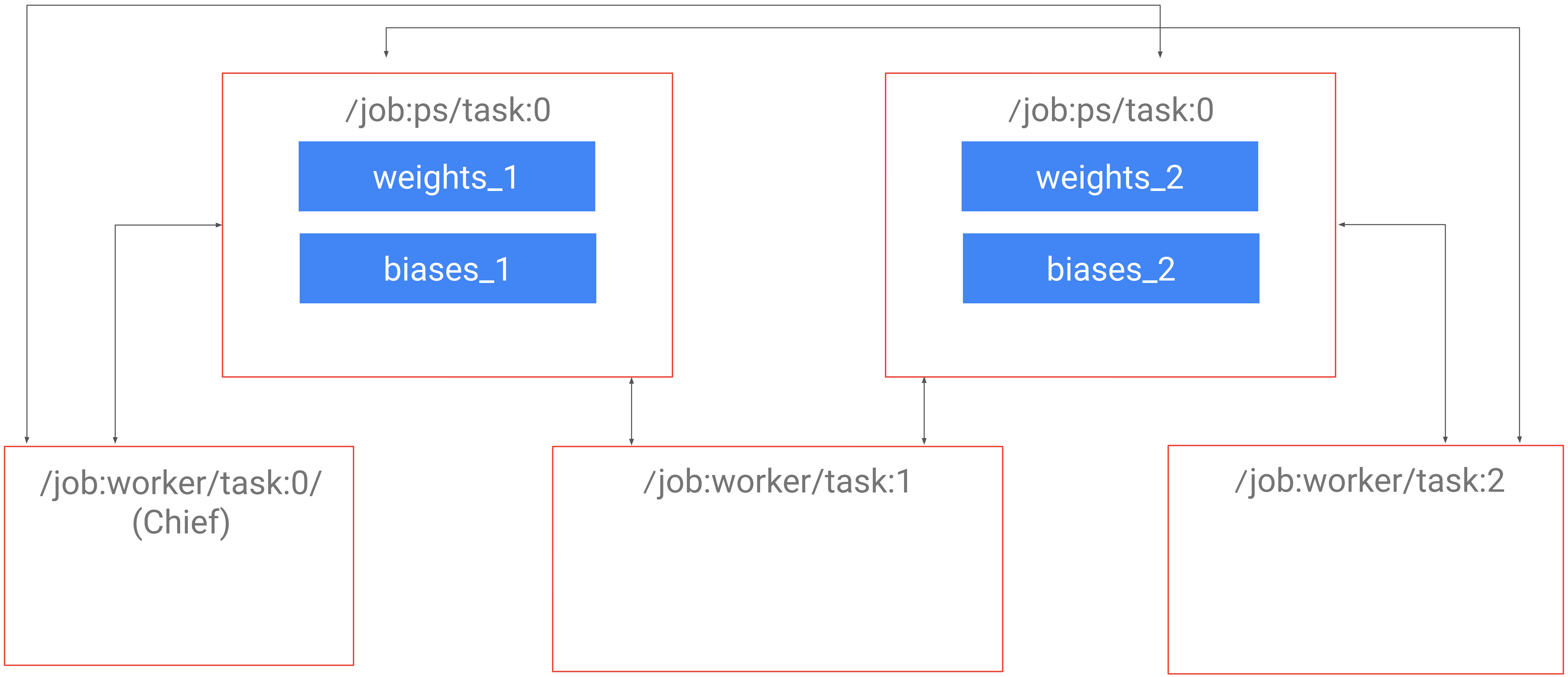
Write to Cloud Storage

# For between-graph replication, make worker 0 the one to write checkpoints

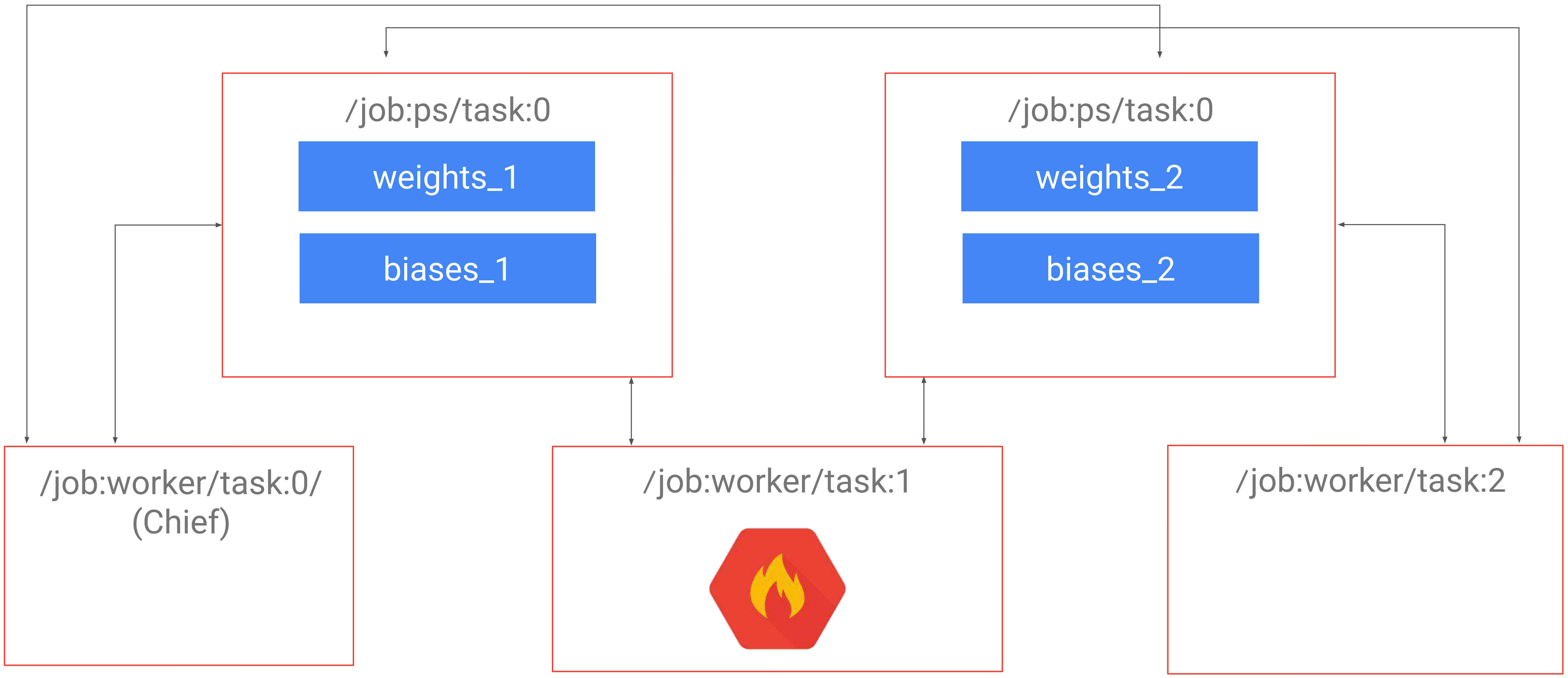
```
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):
  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  # ...

saver = tf.train.Saver(sharded=True)
is_chief = FLAGS.task_index == 0
with tf.Session(server.target) as sess:
  while True:
    # ...
    if is_chief and step % 1000 == 0:
      saver.save(sess, "gs://mybucket/chk_{}".format(step))
```

When a fault occurs,

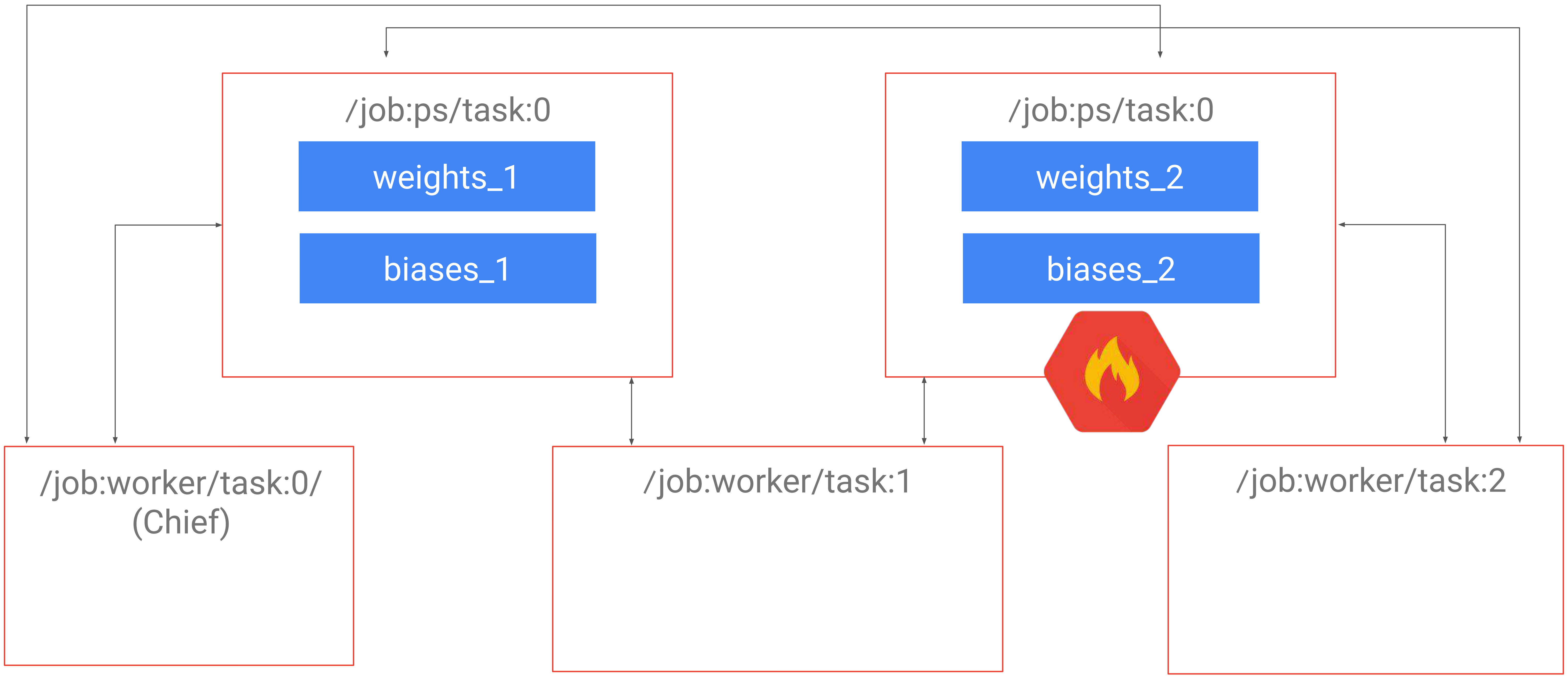
the response is dependent on the failed task



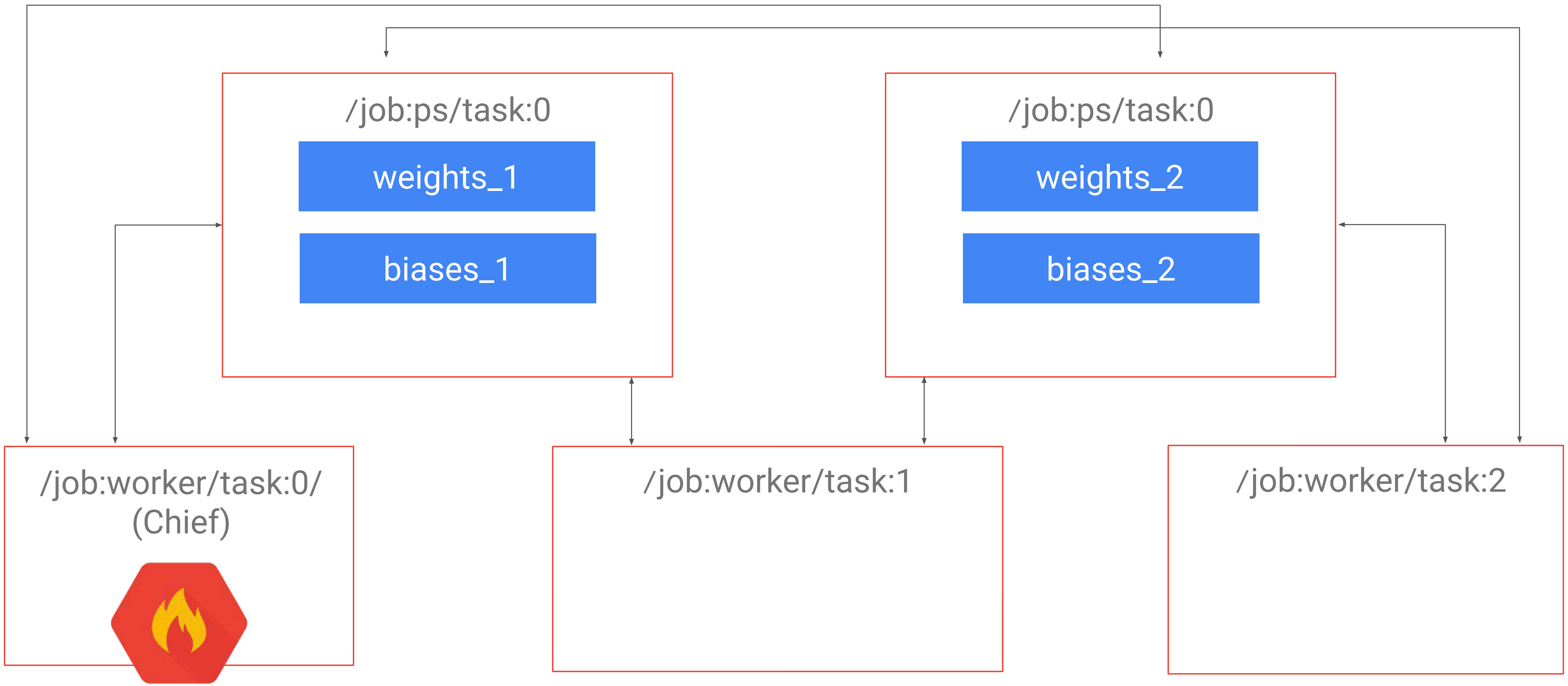
# When a worker fails, it will recover and carry on



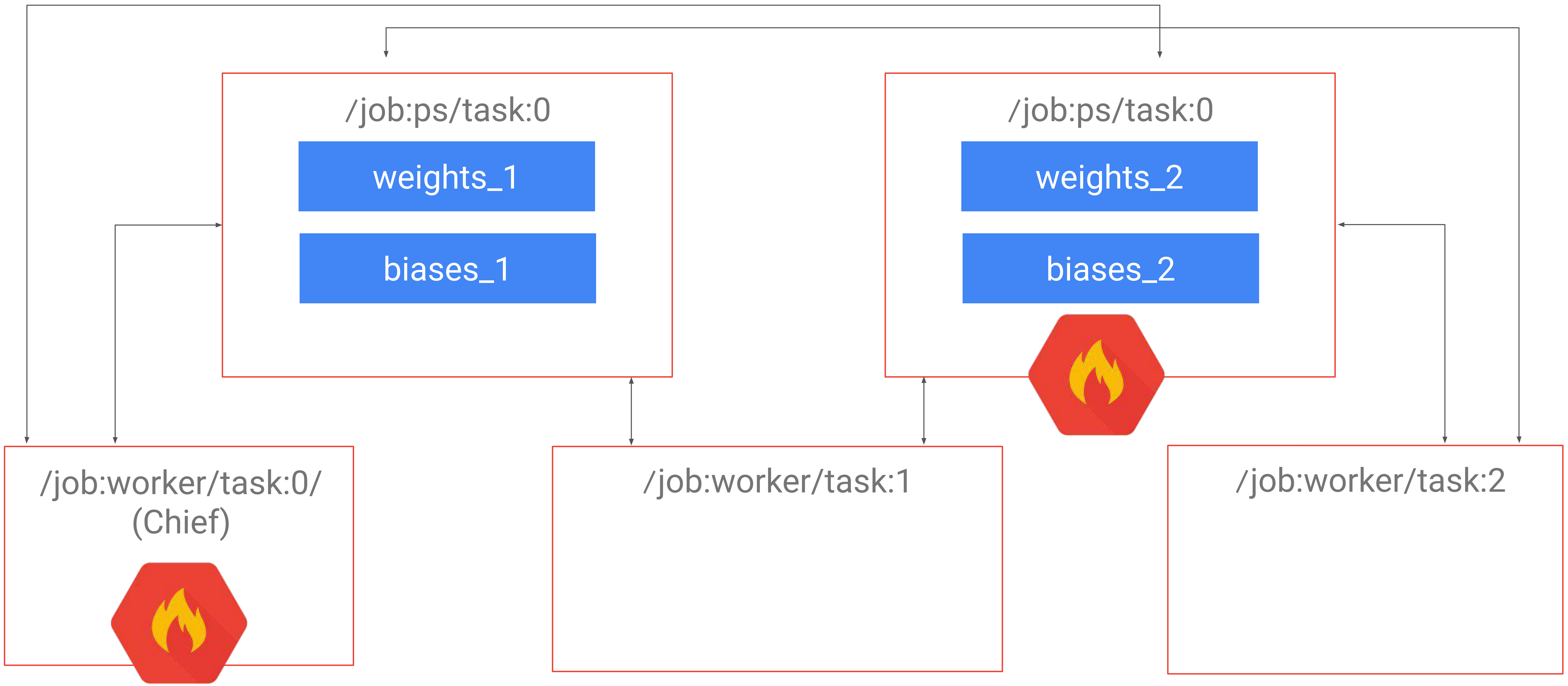
# When a PS task fails, the chief interrupts training and restores PS tasks from the last checkpoint



# When a chief fails, it doesn't know whether a PS task has failed



# So, a chief interrupts training and restores from a checkpoint



# With local training and `tf.Session()`, you have to manage initialization or restoring from a checkpoint

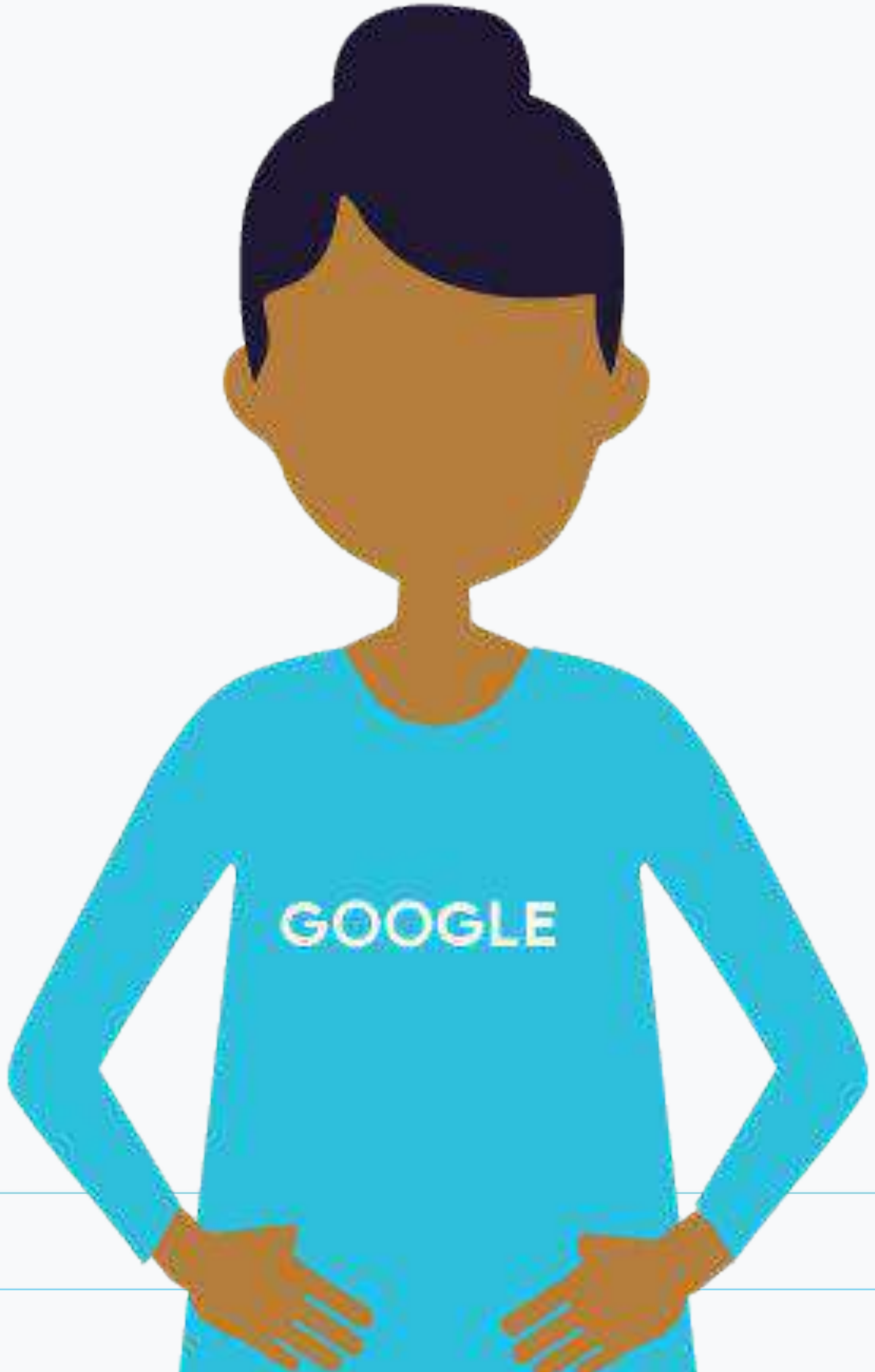
```
# Single-process code.
with tf.Session() as sess:
  sess.run(init_op)  # Or saver.restore(sess, …)
  for _ in range(NUM_STEPS):
    sess.run(train_op)
```

# MonitoredTrainingSession automates recovery process

Automatically initializes and/or restores variables

```
# Distributed code.
server = tf.train.Server(...)
is_chief = FLAGS.task_index == 0
with tf.train.MonitoredTrainingSession(server.target, is_chief) as sess:
  while not sess.should_stop():
    sess.run(train_op)
```

Recovers from PS failures, and can run additional code in hooks





Title Safe >

< Action Safe