TensorFlow

Narayan Hegde, Google

Narayan Hegde
IISc, Bangalore
Maps and Vision Research, Google

Email : hegde@google.com

What is Deep Learning?

What is Deep Learning?

- A powerful class of machine learning model
- Model Human Brain way of learning

What is Deep Learning?

Model:
$y = \mathbf{W}x+\mathbf{b}$

prediction

Parameters

Learner

"cat"

What is Deep Learning?

Model:
Deep Neural Network

prediction

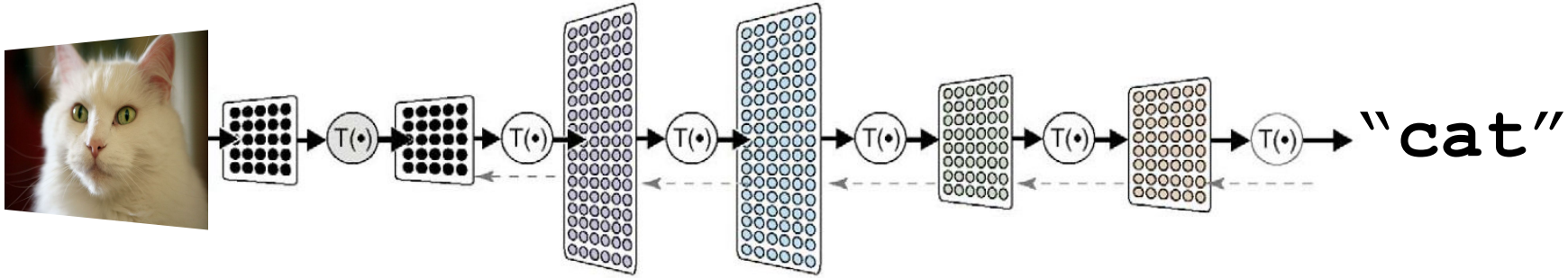
Learner

"cat"

Parameters

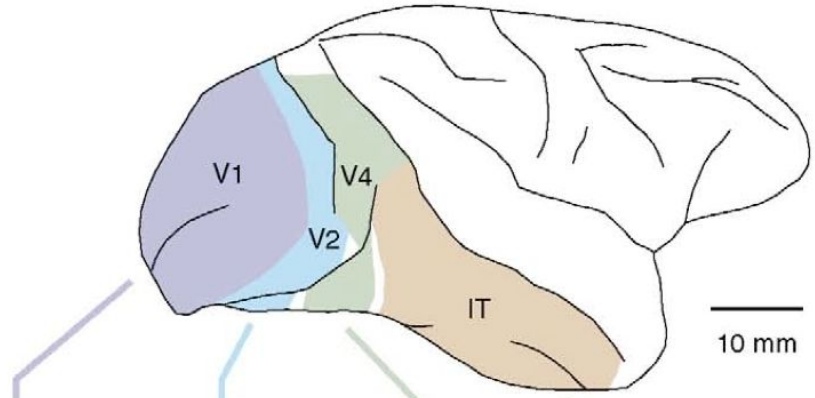
What is Deep Learning?

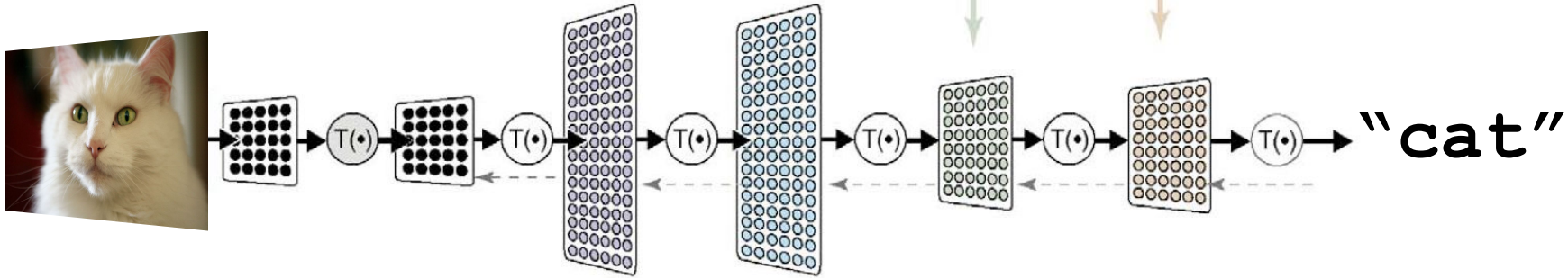
- A powerful class of machine learning model
- Collection of simple, trainable mathematical functions



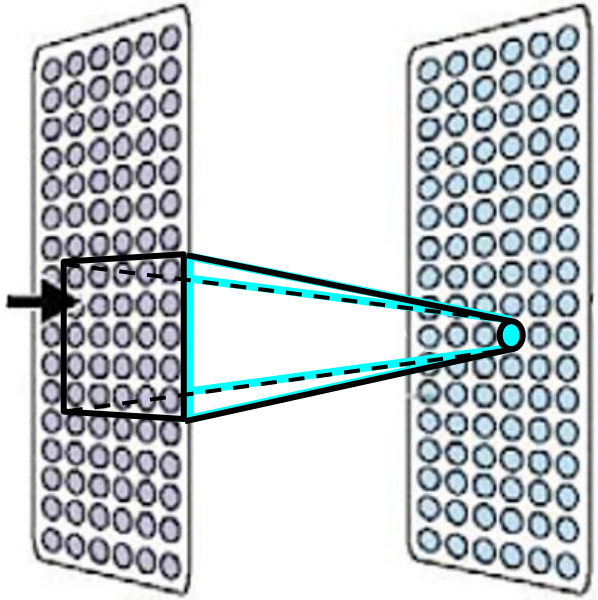
What is Deep Learning?

- Loosely based on (what little) we know about the brain





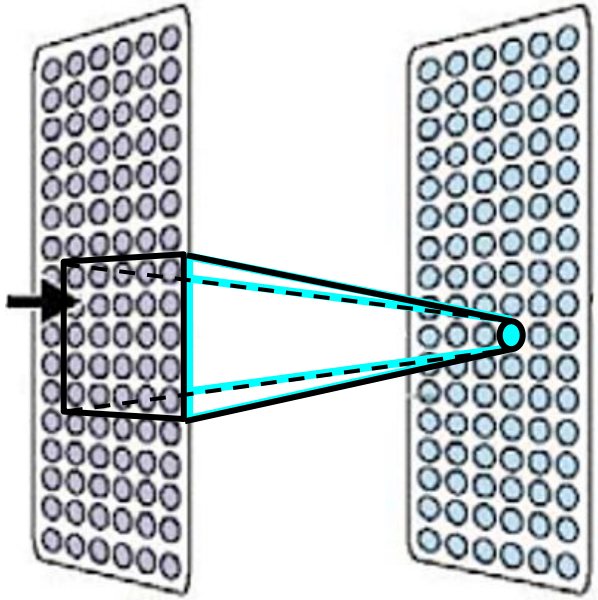
What is Deep Learning?



Commonalities with real brains:

- Each neuron is connected to a small subset of other neurons.
- Neurons learn to cooperate to accomplish the task.

What is Deep Learning?



Each neuron is a relatively simple mathematical function.

$$y = g(\vec{w} \cdot \vec{x} + b)$$

But working together, they can solve complicated tasks.

IS THIS A

CAT or DOG?

CAT DOG

OUTPUT LAYER

ACTIVATED NEURONS

INPUT LAYER

DEEP *neural* NETWORK

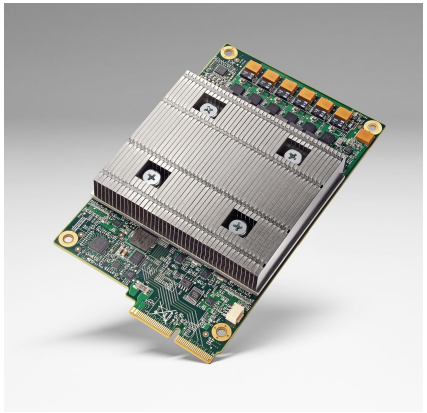
Why now?

- Performance improvements: GPUs & Cloud computing
- Algorithmic improvements
- Larger datasets
- Excellent tools

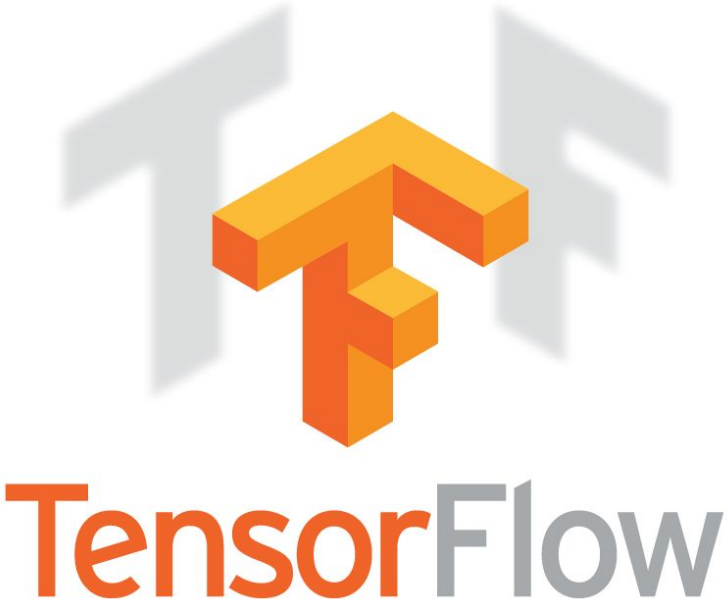
Machine learning had to wait too.

CPU

GPU

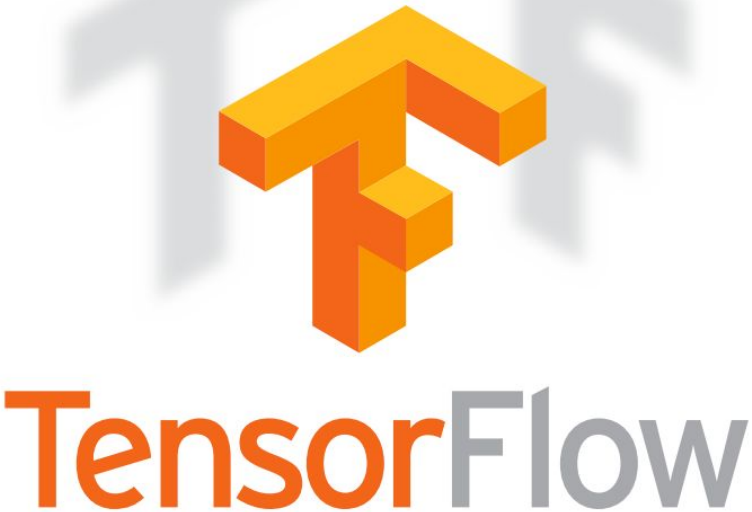


TPU



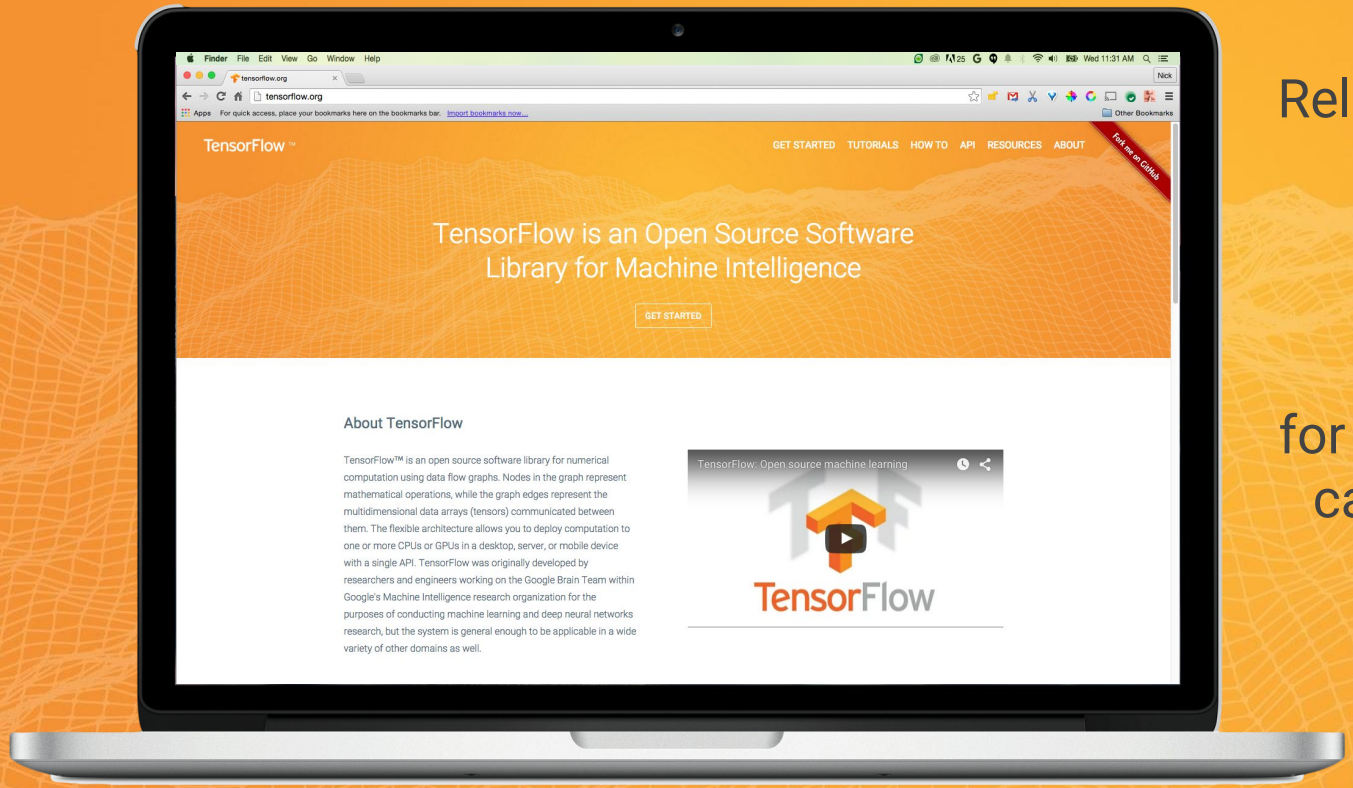
- Fast, flexible, and scalable open-source machine learning library
- For research and production
- Runs on CPU, GPU, Android, iOS, Raspberry Pi, Datacenters
- Apache 2.0 license

https://research.googleblog.com/2016/11/celebrating-tensorflows-first-year.html



http://tensorflow.org/

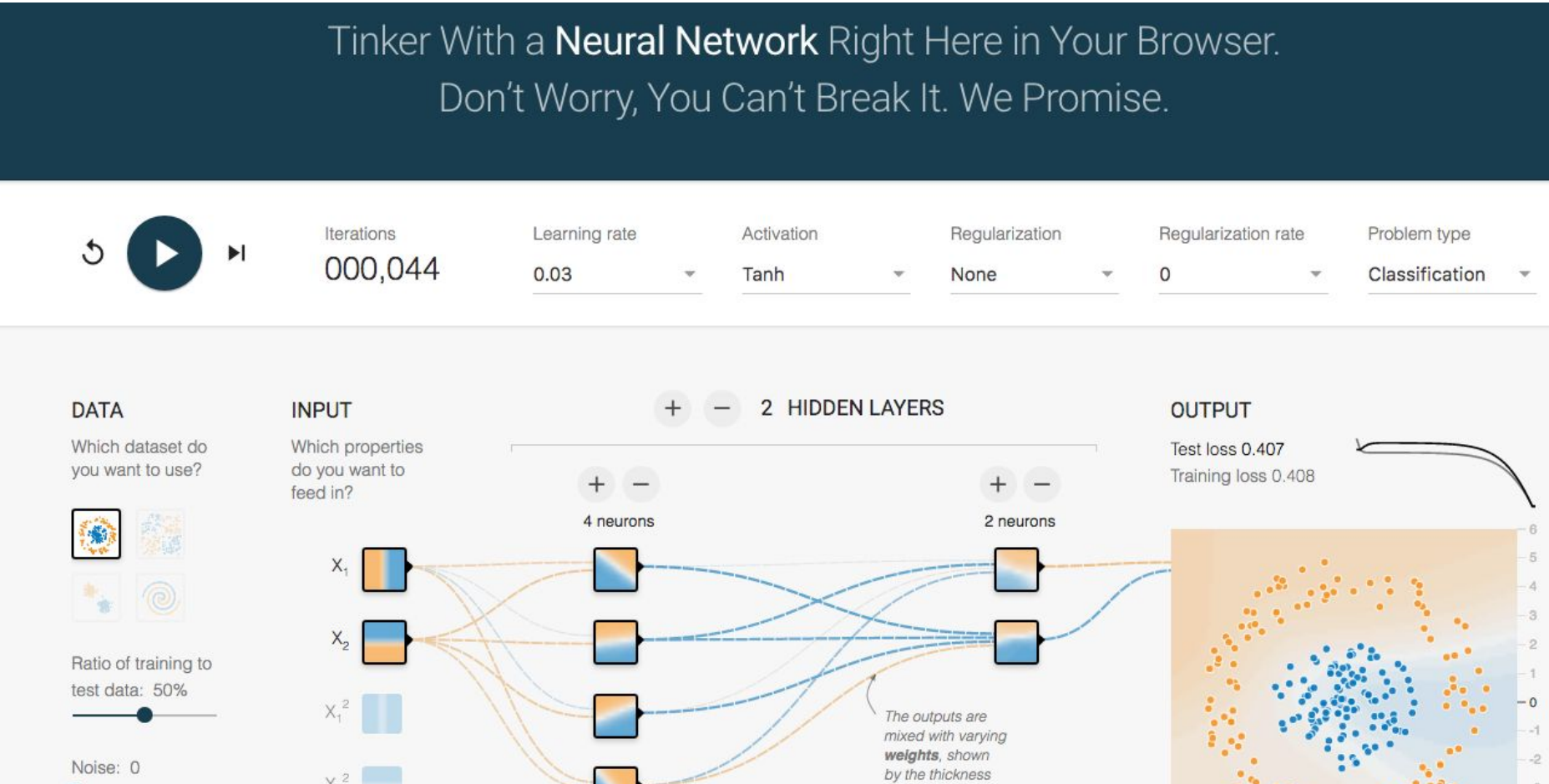
Sharing our tools with researchers and developers around the world

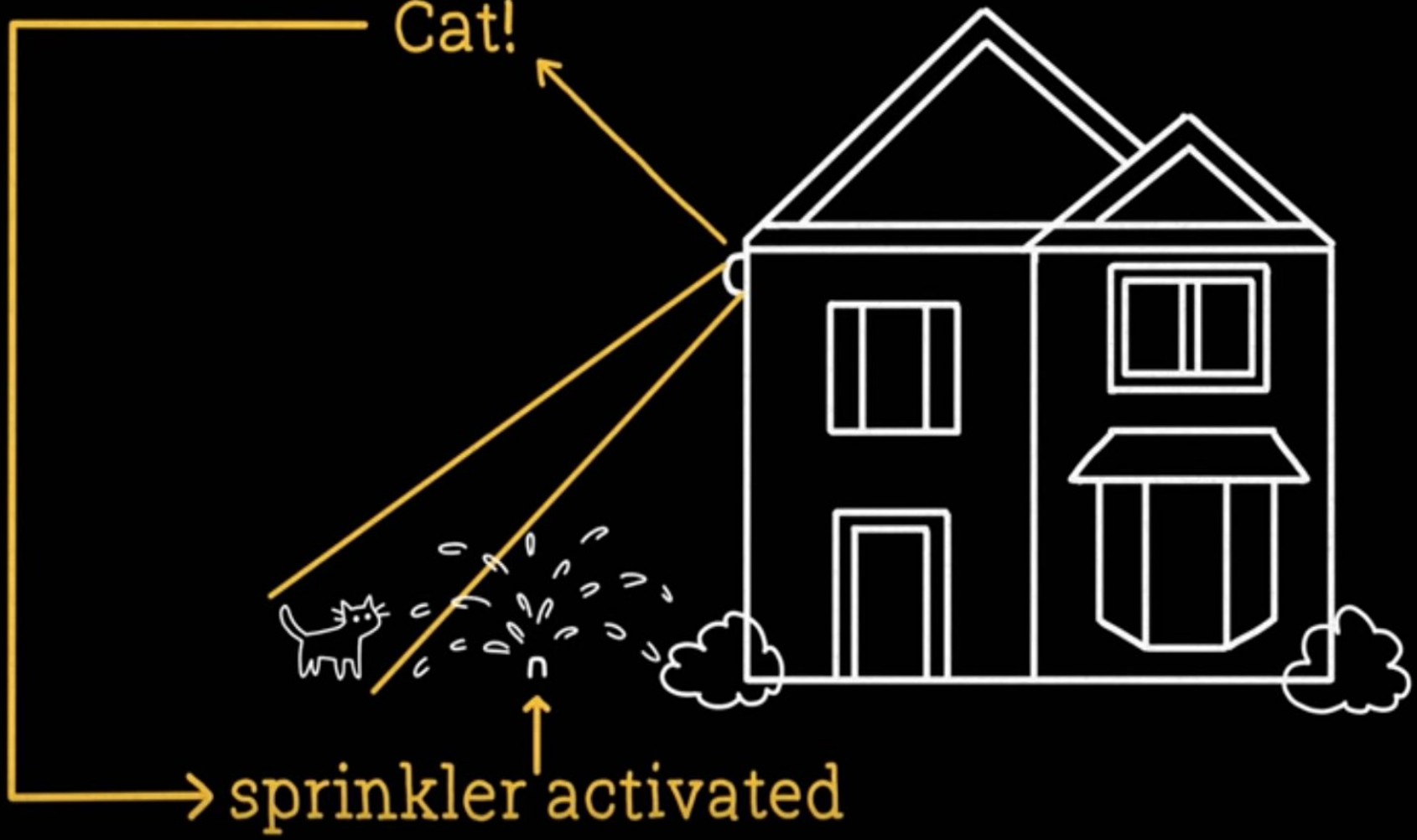


Released in Nov. 2015

#1 repository for “machine learning” category on GitHub

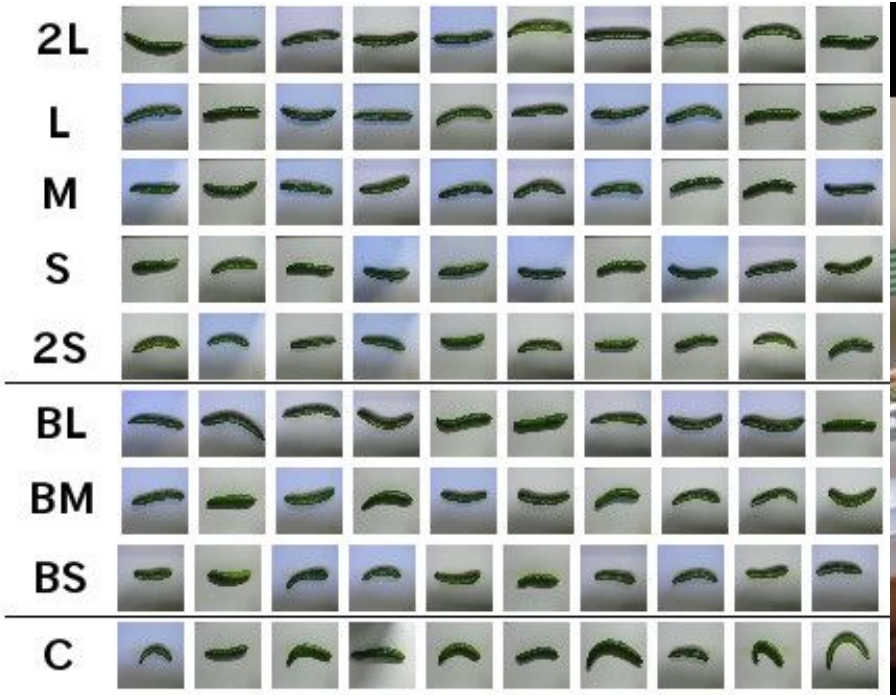
http://playground.tensorflow.org/




Cat!
sprinkler activated

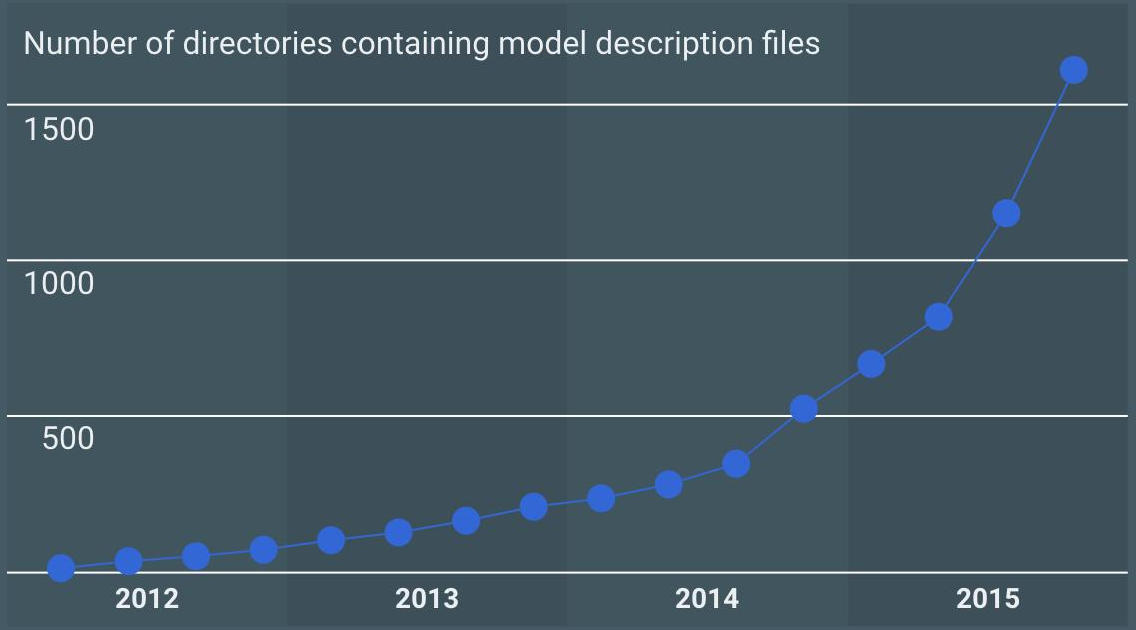
https://cloud.google.com/blog/big-data/2016/08/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow

TensorFlow powered Cucumber Sorter



From: http://workpiles.com/2016/02/tensorflow-cnn-cucumber/

Growing use of deep learning at Google



Across many areas

- AlphaGo
- Apps
- Maps
- Photos
- Gmail
- Speech
- Android
- YouTube
- Translation
- Robotics Research
- Image Understanding
- Natural Language Understanding
- Drug Discovery

Plenty of raw data

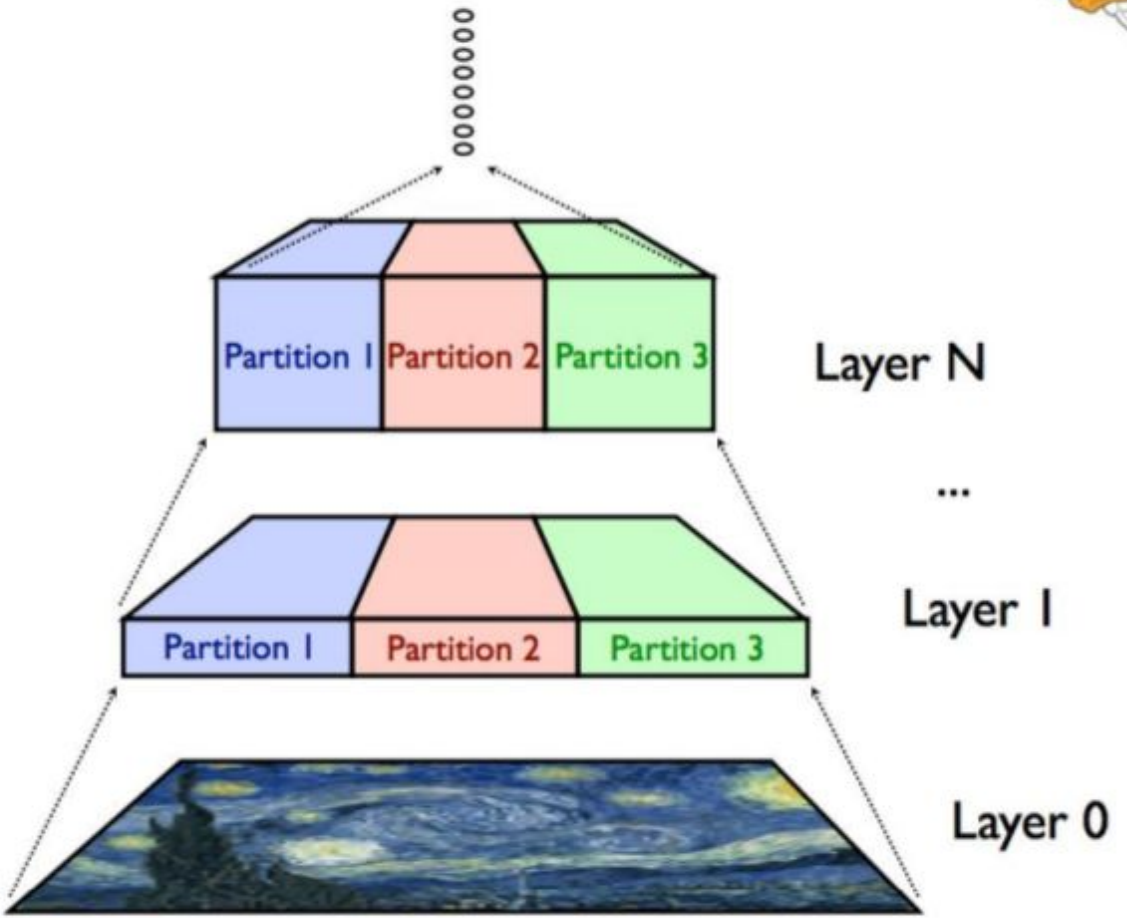
- **Text**: trillions of words of English + other languages
- **Visual data**: billions of images and videos
- **Audio**: tens of thousands of hours of speech per day
- **User activity**: queries, marking messages spam, etc.
- **Knowledge graph**: billions of labelled relation triples
- ...

How can we build systems that truly understand this data?

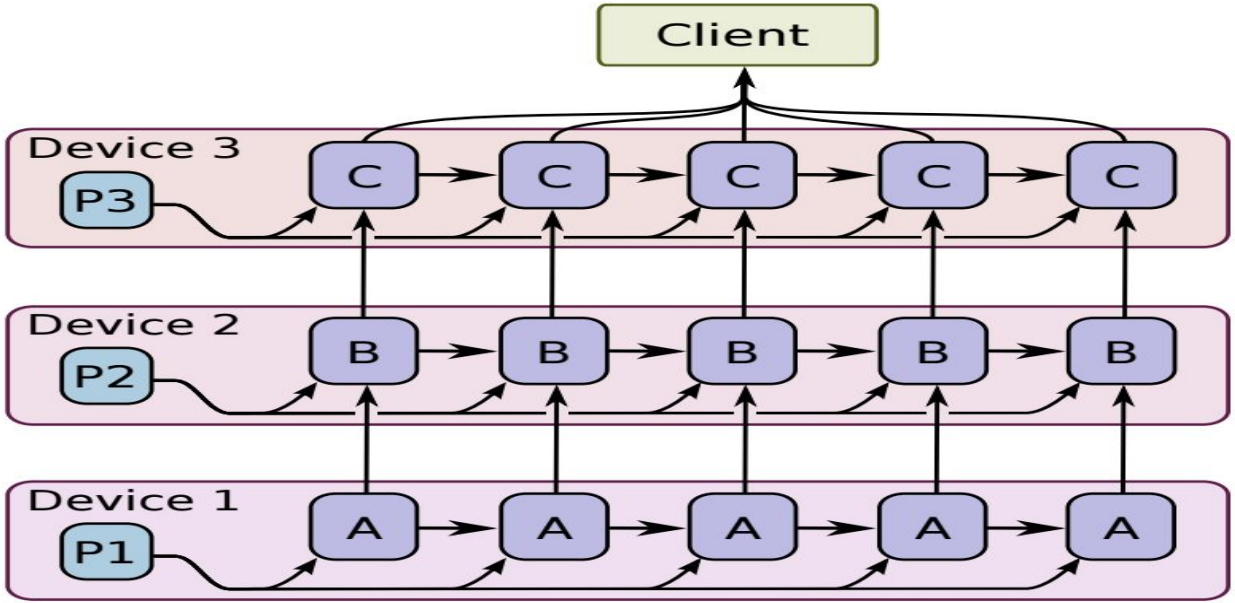
How Can We Train Large, Powerful Models Quickly?

- Exploit many kinds of parallelism
  - Model parallelism
  - Data parallelism

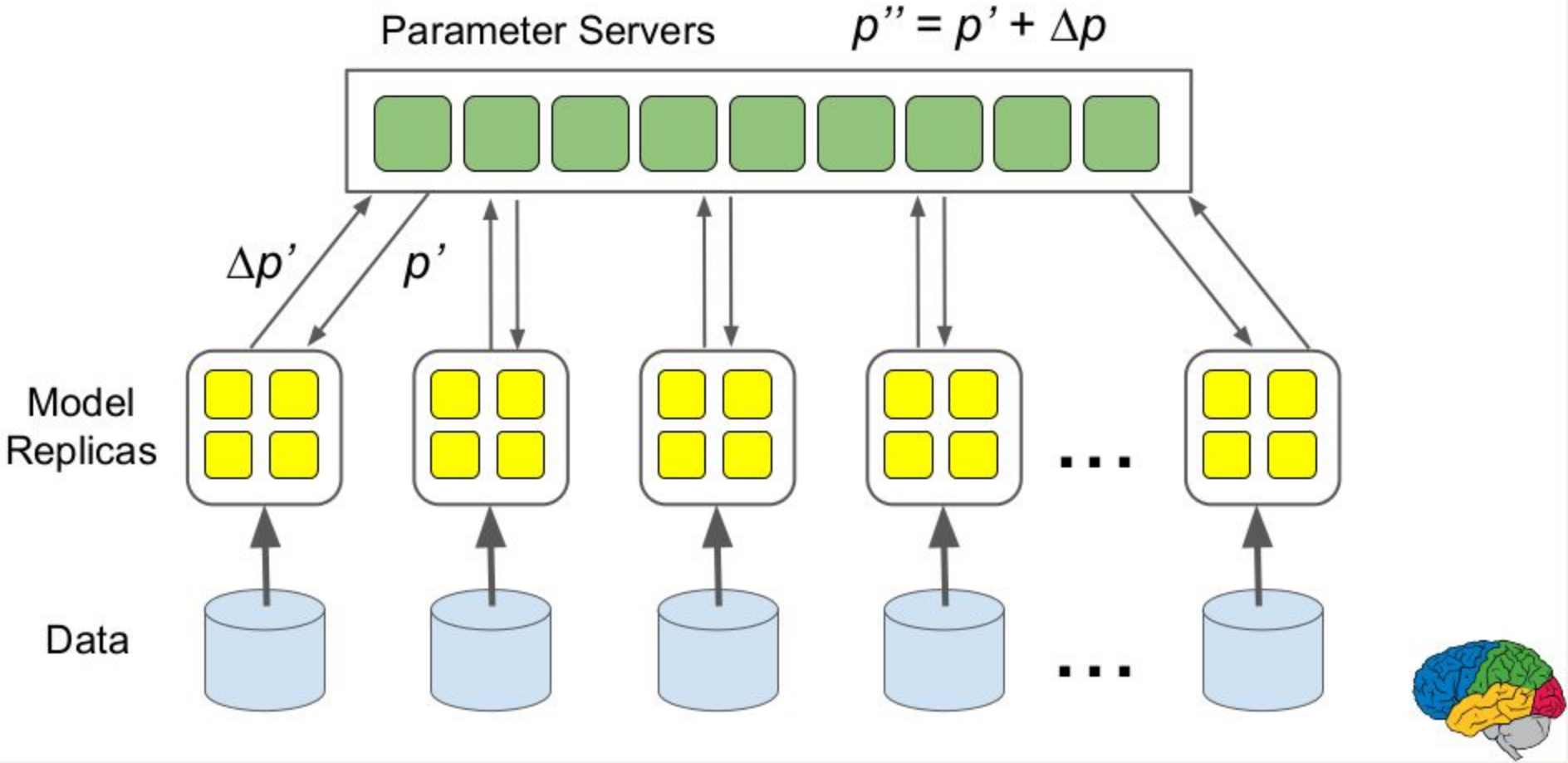
Model Parallelism: Partition model across machines

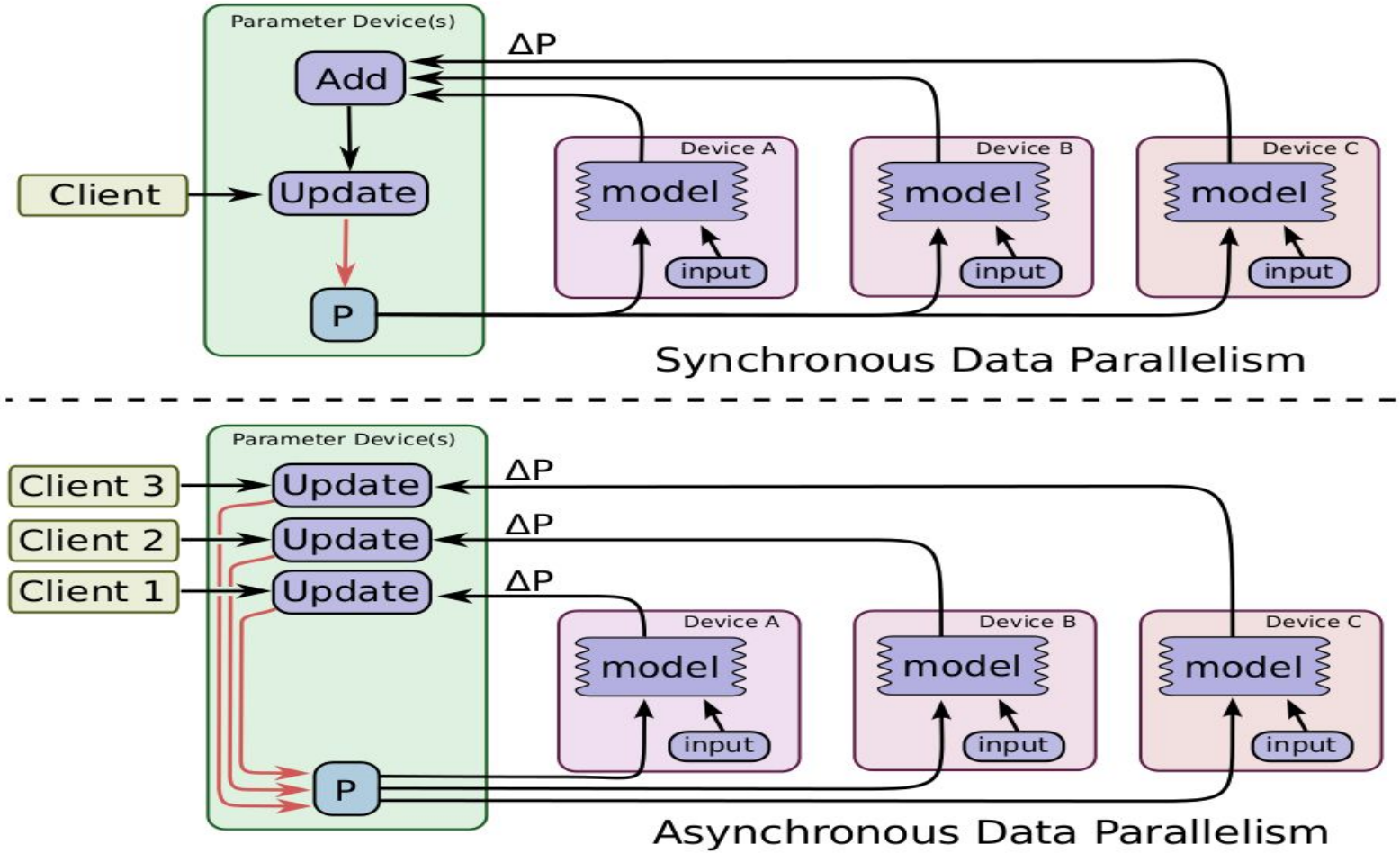


Model Parallelism



Data Parallelism




Parameter Device(s)
Add
Client
Update
P
ΔP
Device A
model
input
Device B
model
input
Device C
model
input
Synchronous Data Parallelism
Parameter Device(s)
Client 3
Client 2
Client 1
Update
Update
Update
P
ΔP
ΔP
ΔP
Device A
model
input
Device B
model
input
Device C
model
input
Asynchronous Data Parallelism

Data Parallelism Choices

Can do this **synchronously**:

- N replicas eqivalent to an N times larger batch size
- Pro: No noise
- Con: Less fault tolerant (requires recovery if any single machine fails)

Can do this **asynchronously**:

- Con: Noise in gradients
- Pro: Relatively fault tolerant (failure in model replica doesn't block other replicas)

(Or **hybrid**: M asynchronous groups of N synchronous replicas)

What is TensorFlow?

From the whitepaper: “TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms.”

In short: TensorFlow is Theano++.

- Symbolic ML dataflow framework that compiles to native / GPU code
- From personal experience: offers drastic reduction in development time

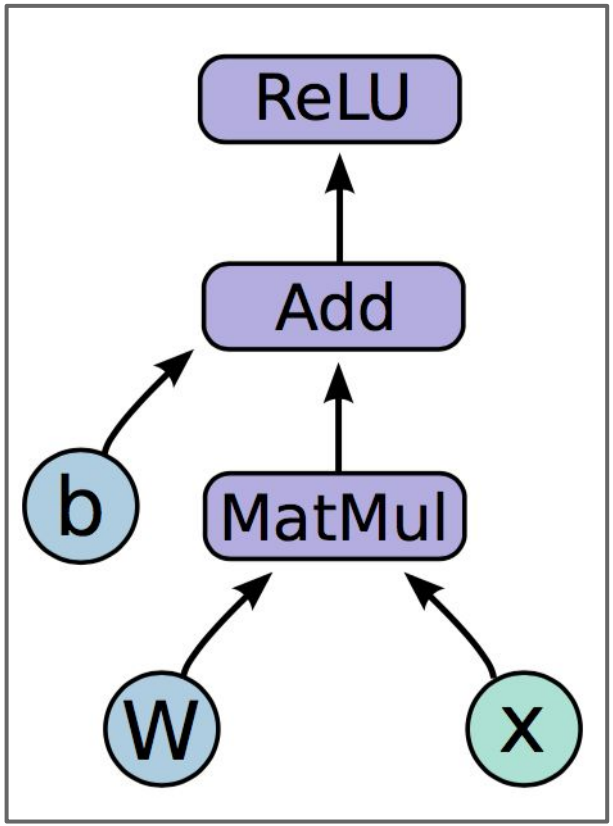
Programming model

Big idea: Express a numeric computation as a **graph.**

- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes

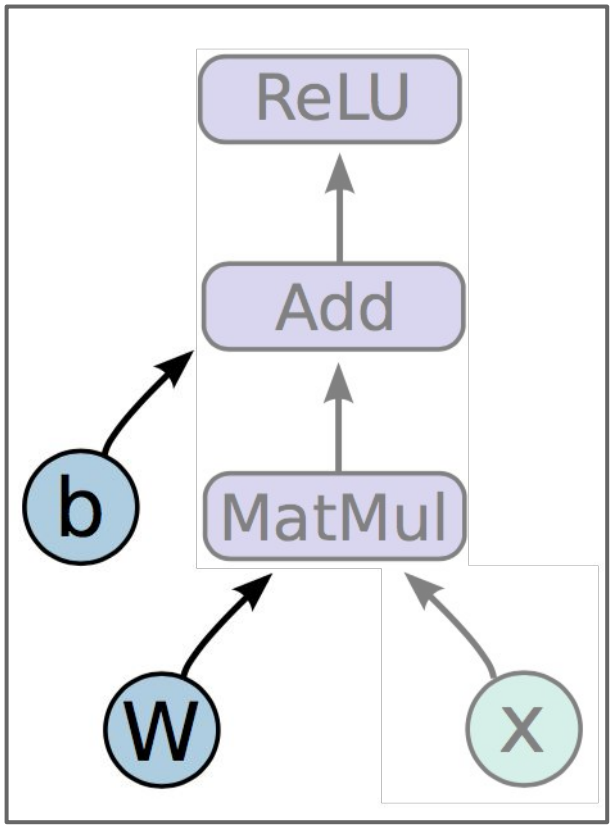
Programming model: NN feedforward

$$h_i = \mathrm{ReLU}(Wx + b)$$



Programming model: NN feedforward

$$h_i = \mathrm{ReLU}(Wx + b)$$



Variables are 0-ary stateful nodes which output their current value.
(State is retained across multiple executions of a graph.)

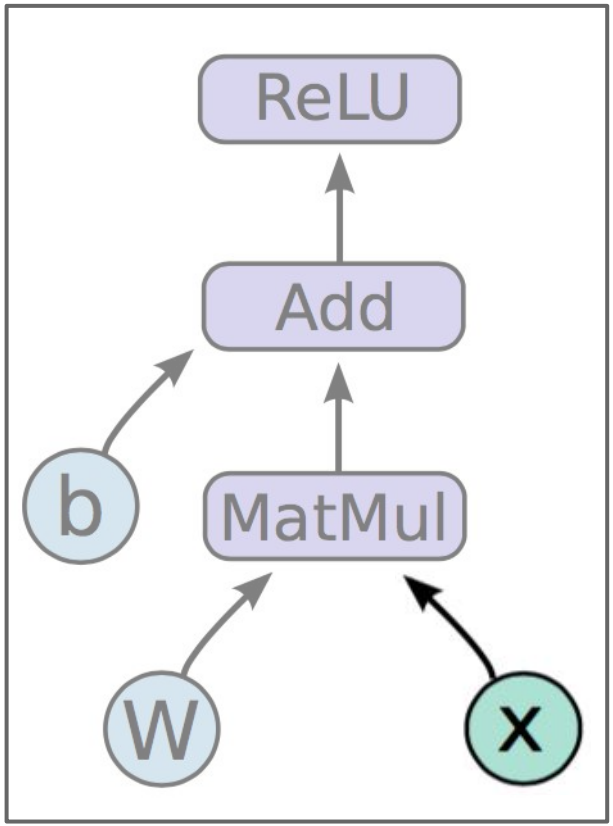
(parameters, gradient stores, eligibility traces, …)

Programming model: NN feedforward

$$h_i = \mathrm{ReLU}(Wx + b)$$

Placeholders are 0-ary nodes whose value is fed in at execution time.

(inputs, variable learning rates, …)



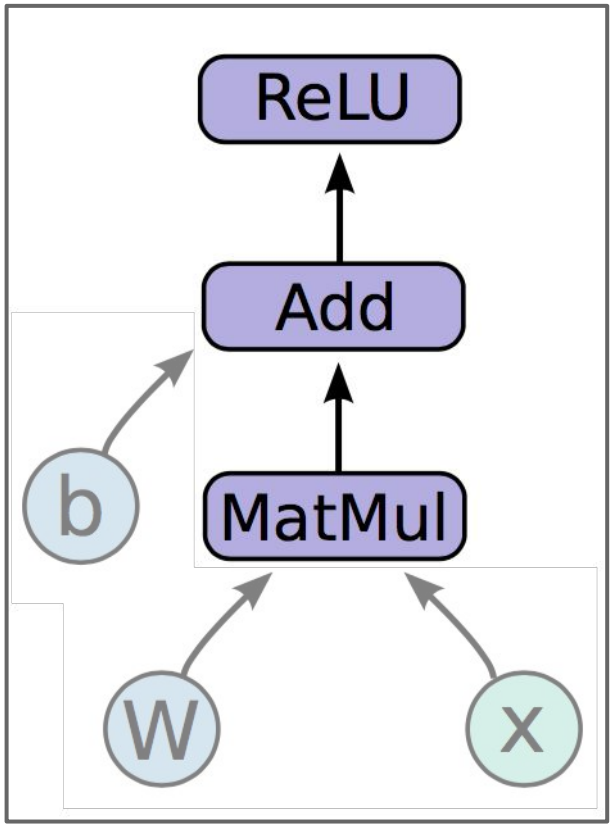
Programming Model and Basic Concepts

- Variables
- Operations and Kernels
- Sessions

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Programming model: NN feedforward

$$h_i = \mathrm{ReLU}(Wx + b)$$



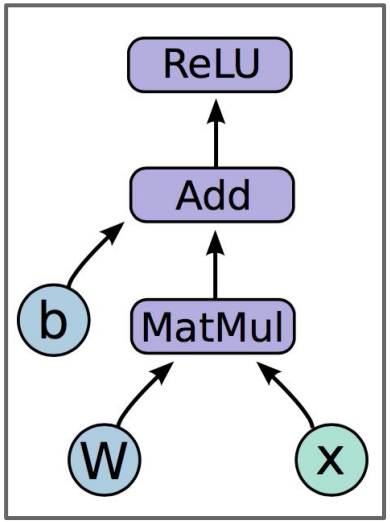
Mathematical operations:

MatMul: Multiply two matrix values.
Add: Add elementwise (with broadcasting).
ReLU: Activate with elementwise rectified linear function.

In code, please!

```
import tensorflow as tf

1  b = tf.Variable(tf.zeros((100,)))
   W = tf.Variable(tf.random_uniform((784, 100),
                                     -1, 1))

2  x = tf.placeholder(tf.float32, (None, 784))
3  h_i = tf.nn.relu(tf.matmul(x, W) + b)
```

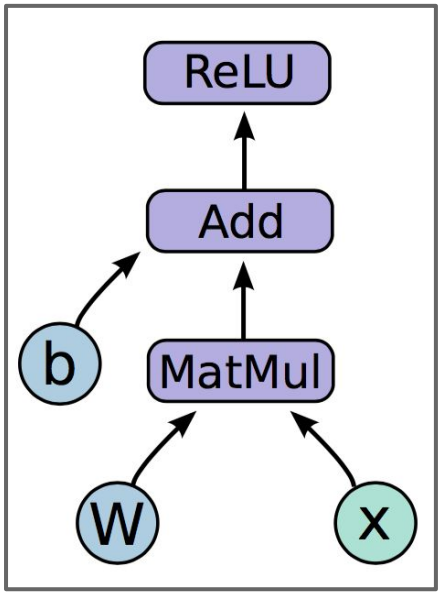
$$h_i = \mathrm{ReLU}(Wx + b)$$



How do we run it?

So far we have defined a **graph**.

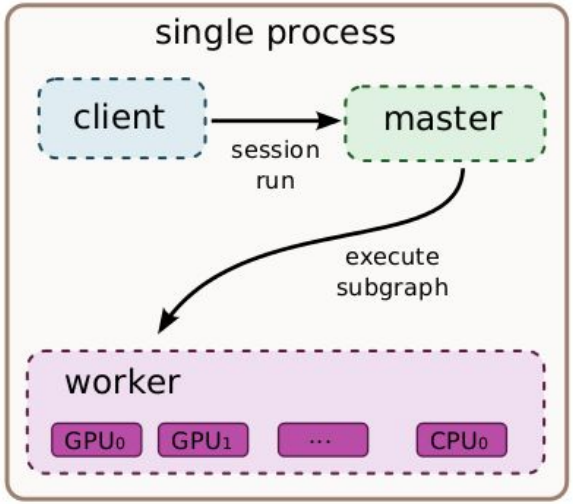
We can deploy this graph with a **session**: a binding to a particular execution context (e.g. CPU, GPU)

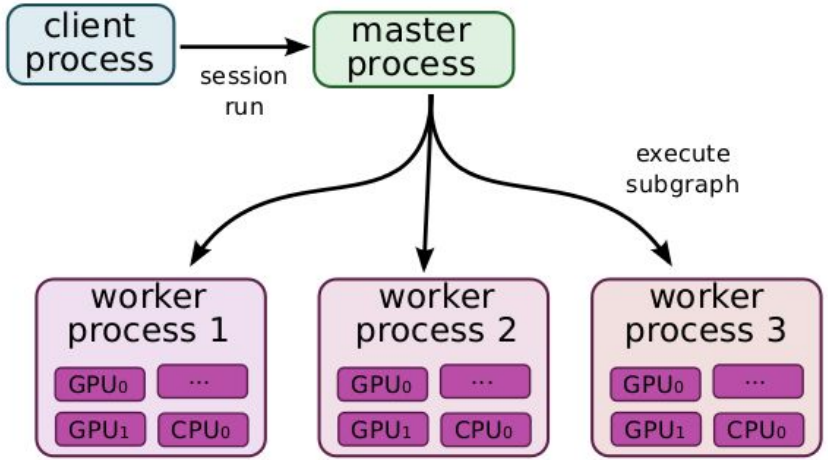


Implementation

- Single-Device Execution
- Multi-Device Execution
  - Node Placement

Single machine and distributed system structure

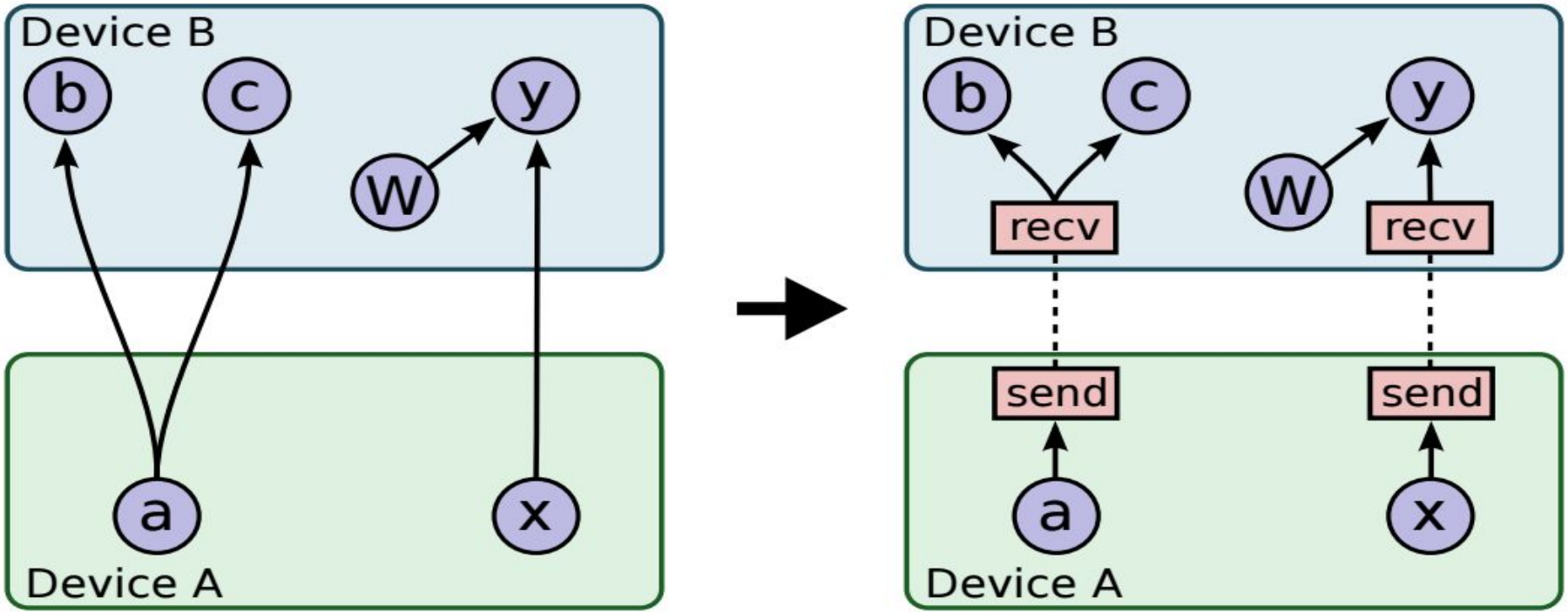




Implementation

- Single-Device Execution
- Multi-Device Execution
  - Node Placement
  - Cross-Device Communication

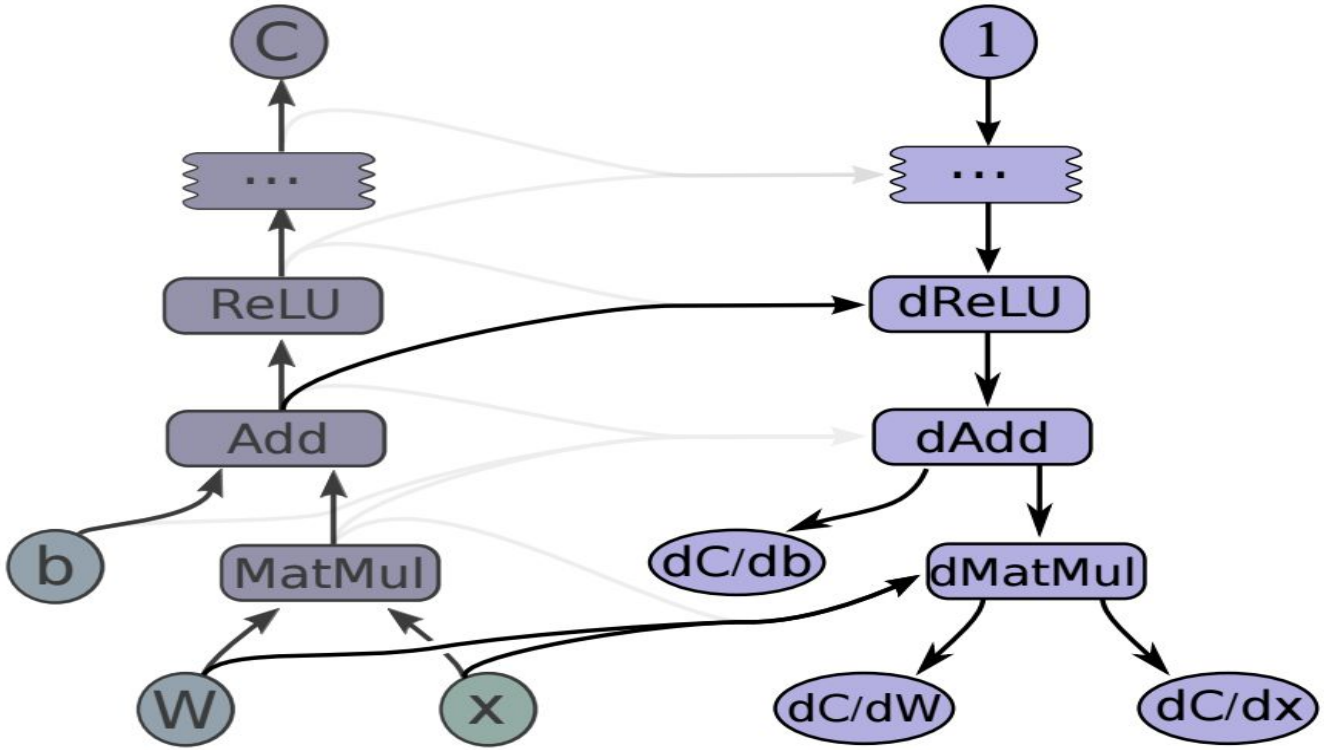
Cross-Device Communication



Implementation

- Single-Device Execution
- Multi-Device Execution
  - Node Placement
  - Cross-Device Communication
- Distributed Execution
  - Fault Tolerance
  - Checkpoint restoration

Gradient Computation



Basic flow

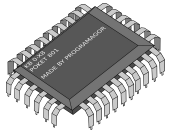
1. Build a graph
    a. Graph contains parameter specifications, model architecture, optimization process, …
    b. Somewhere between 5 and 5000 lines
2. Initialize a session
3. Fetch and feed data with Session.run
    a. Compilation, optimization, etc. happens at this step — you probably won’t notice

Distributed TensorFlow

Client

/job:worker/task:0/

cpu:0



gpu:0

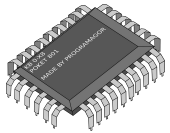
Distributed TensorFlow

```
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



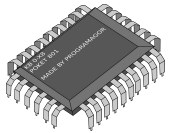
Distributed TensorFlow

```
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



DMA

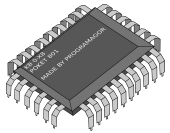
Distributed TensorFlow

```
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

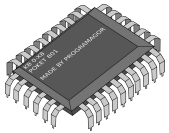
/job:worker/task:0/

cpu:0 gpu:0



/job:ps/task:0/

cpu:0



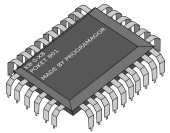
Distributed TensorFlow

```
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

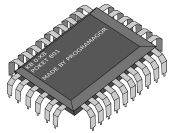
/job:worker/task:0/

cpu:0 gpu:0



/job:ps/task:0/

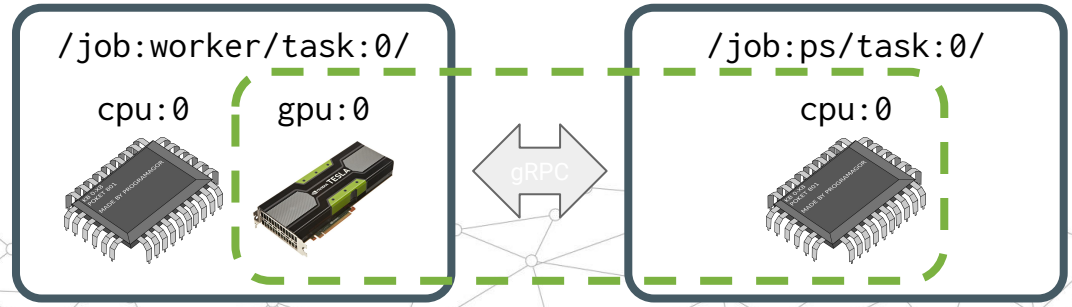
cpu:0



Distributed TensorFlow

```
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)
```

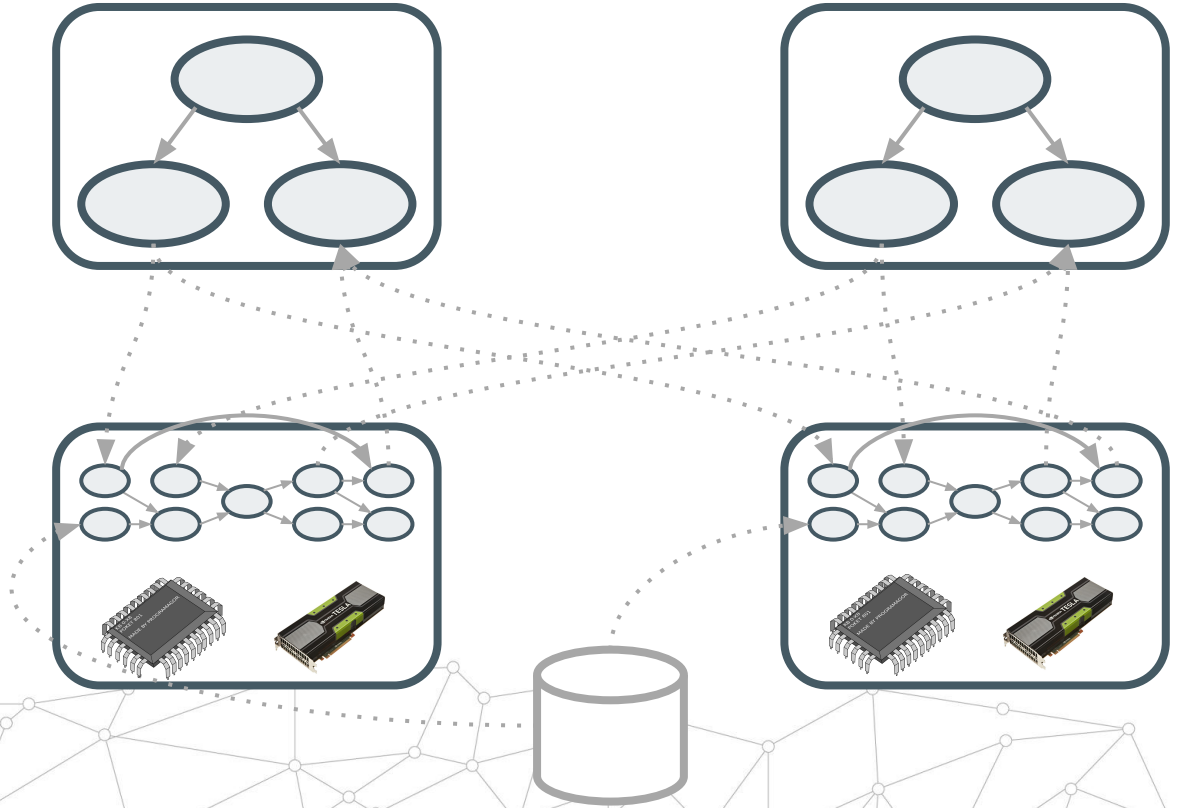
Client



Distributed device placement

PS tasks

Worker tasks

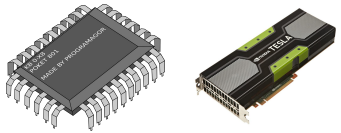


In-graph replication

Client

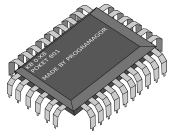
/job:worker/task:0/

cpu:0 gpu:0



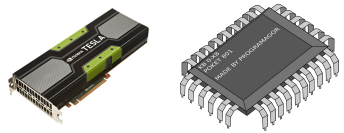
/job:ps/task:0/

cpu:0



/job:worker/task:1/

gpu:0 cpu:0



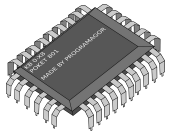
In-graph replication

```
inputs = tf.split(0, num_workers, input)
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
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)
```

Client

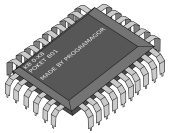
/job:worker/task:0/

cpu:0 gpu:0



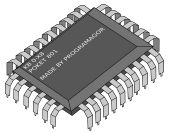

/job:ps/task:0/

cpu:0



/job:worker/task:1/

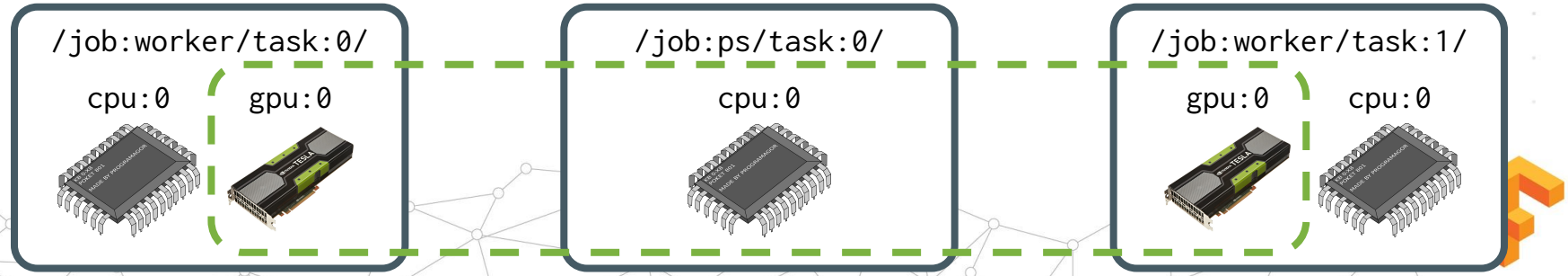
gpu:0 cpu:0

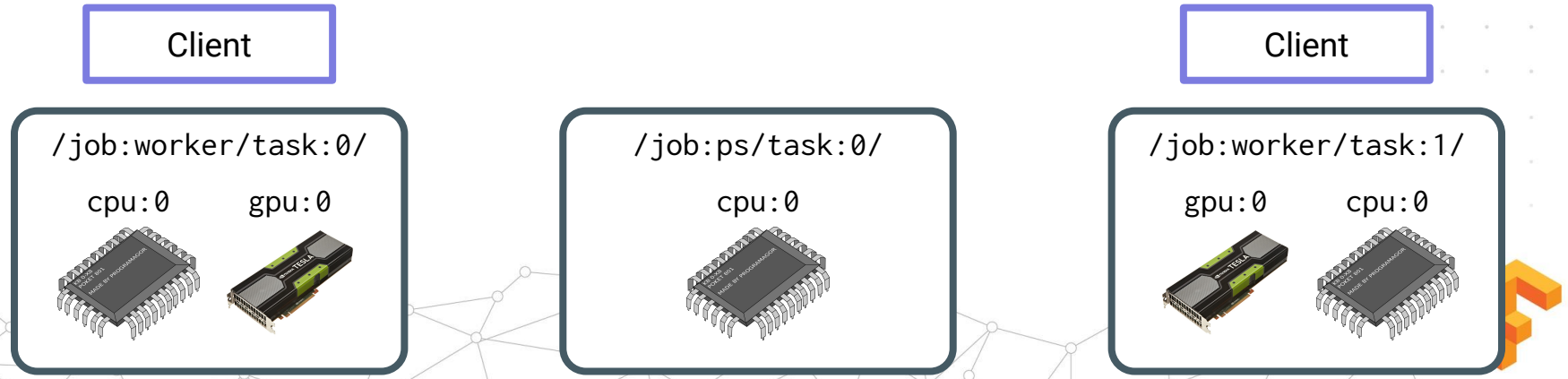
In-graph replication

```
inputs = tf.split(0, num_workers, input)
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
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)
```

Client



Between-graph replication



Between-graph replication

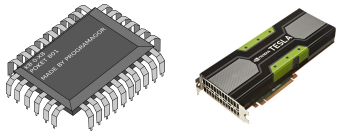
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

Client

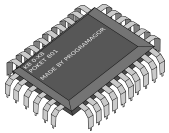
/job:worker/task:0/

cpu:0 gpu:0



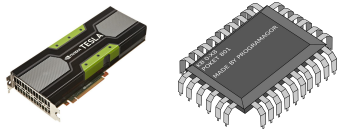
/job:ps/task:0/

cpu:0



/job:worker/task:1/

gpu:0 cpu:0



Between-graph replication

```
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)
```

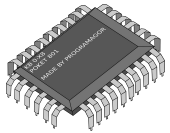
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

Client

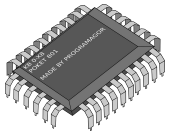
/job:worker/task:0/

cpu:0 gpu:0



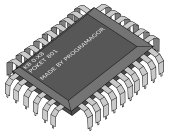
/job:ps/task:0/

cpu:0



/job:worker/task:1/

gpu:0 cpu:0



Between-graph replication

```
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)
```

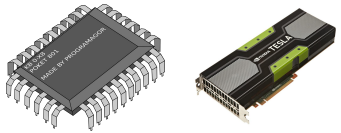
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Client

Client

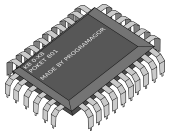
/job:worker/task:0/

cpu:0 gpu:0



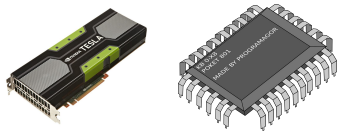
/job:ps/task:0/

cpu:0



/job:worker/task:1/

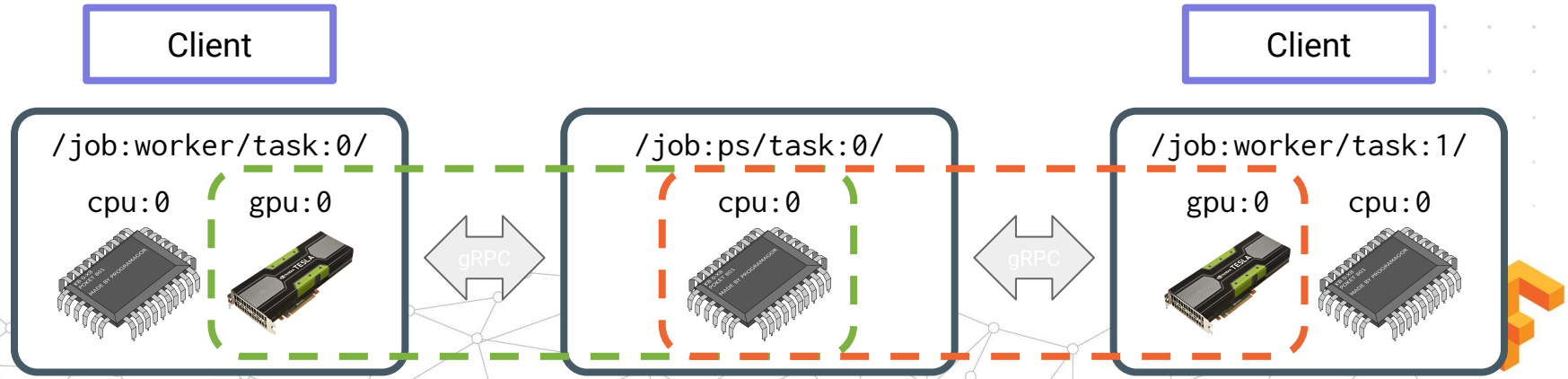
gpu:0 cpu:0



Between-graph replication

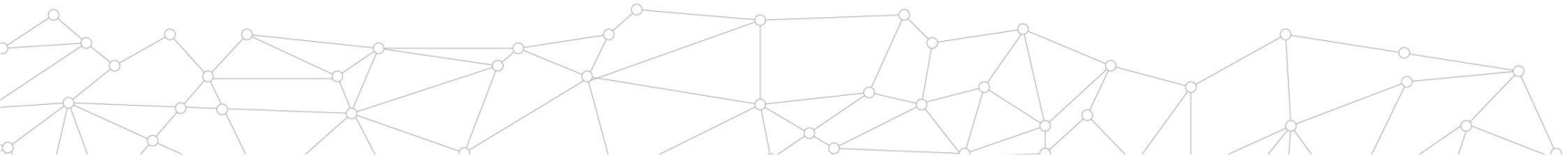
```
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

```
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)
```



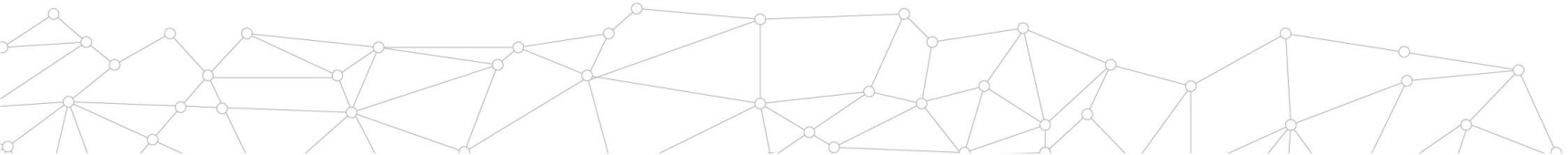
Device placement for variables

```
with tf.device(...):

  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])
```



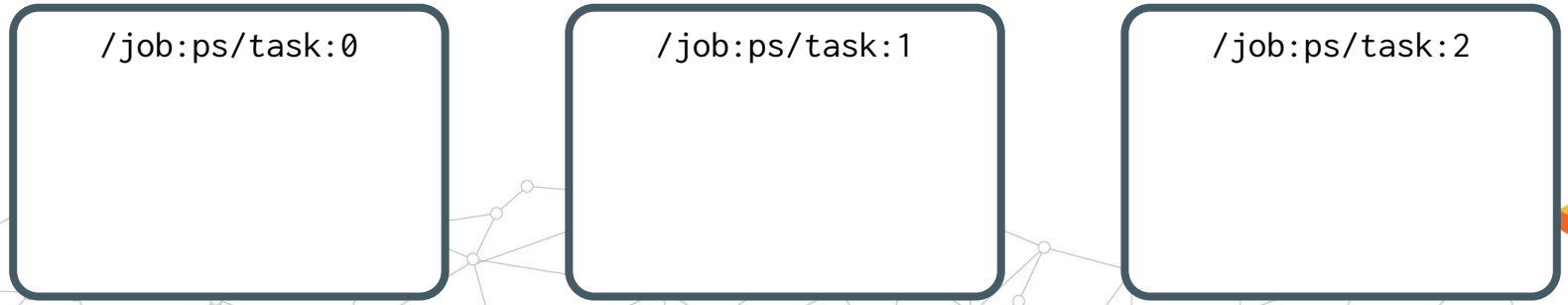
Device placement for variables

```
with tf.device("/job:ps/task:0"):

  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])
```



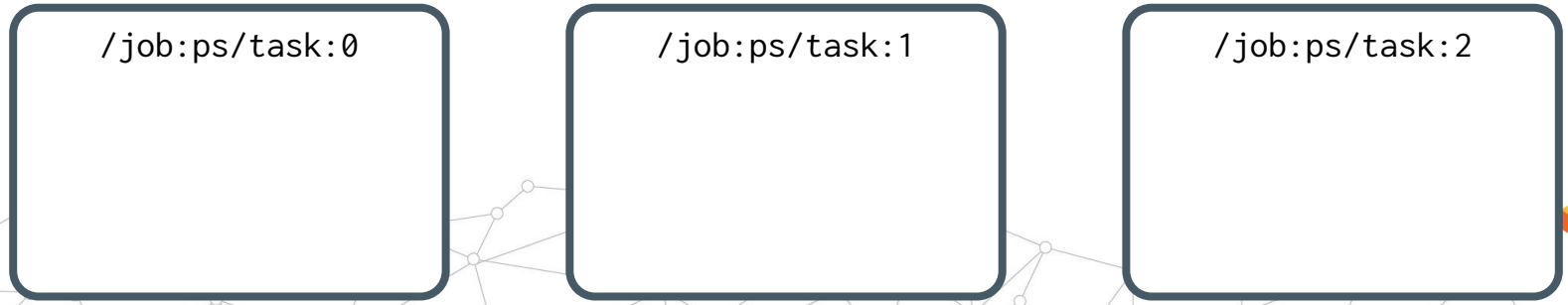
Device placement for variables

```
with tf.device("/job:ps/task:0"):

  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])
```



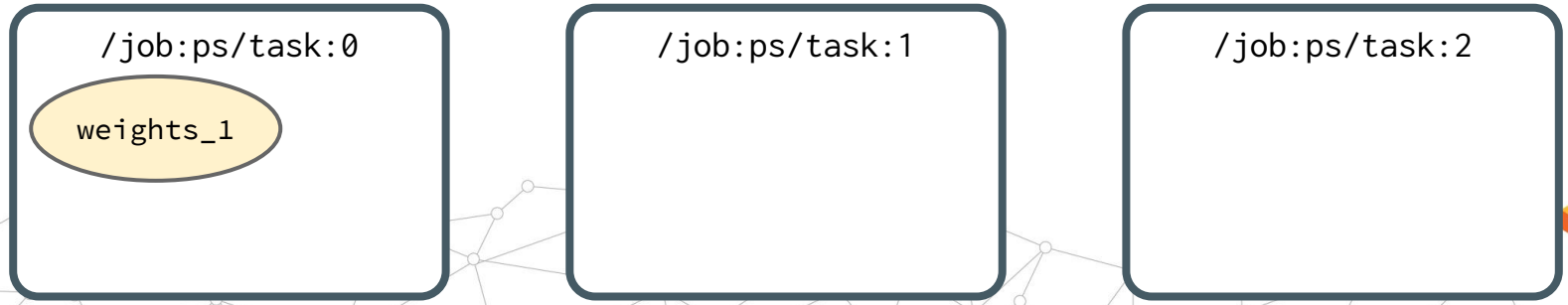
Round-robin variables

```
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])
```



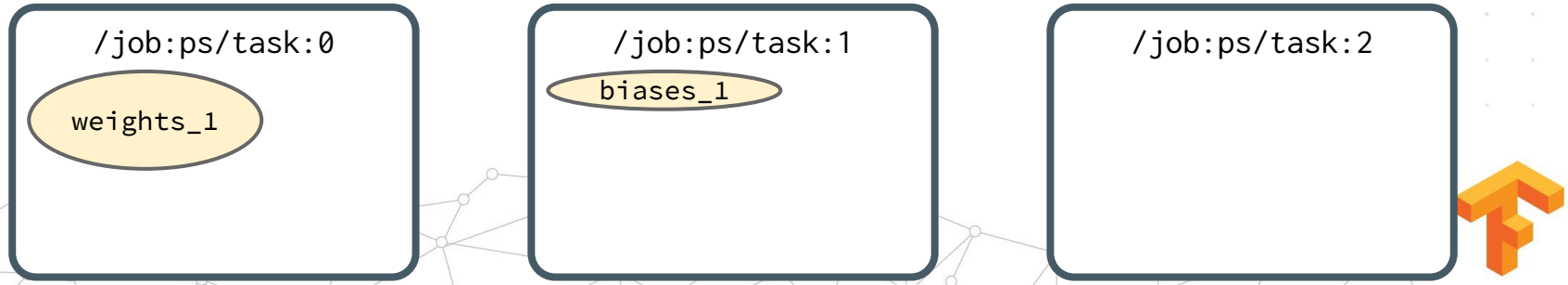
Round-robin variables

```
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])
```



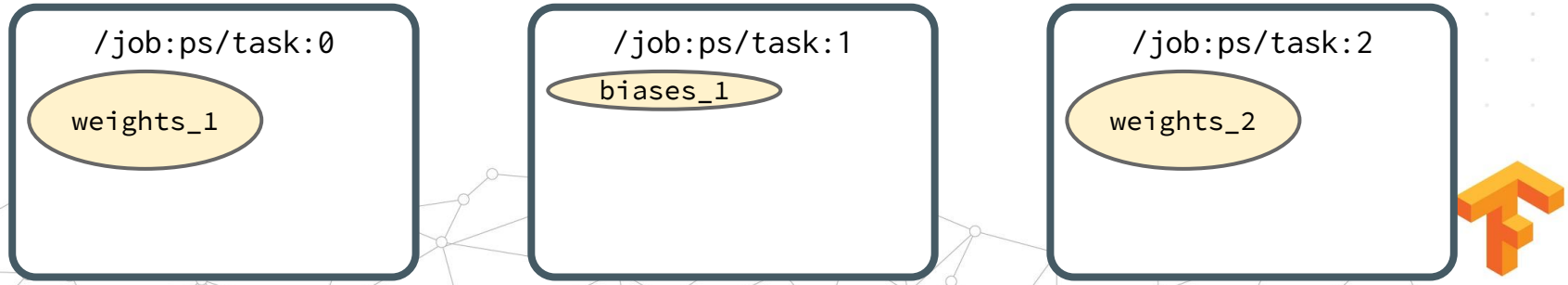
Round-robin variables

```
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])
```



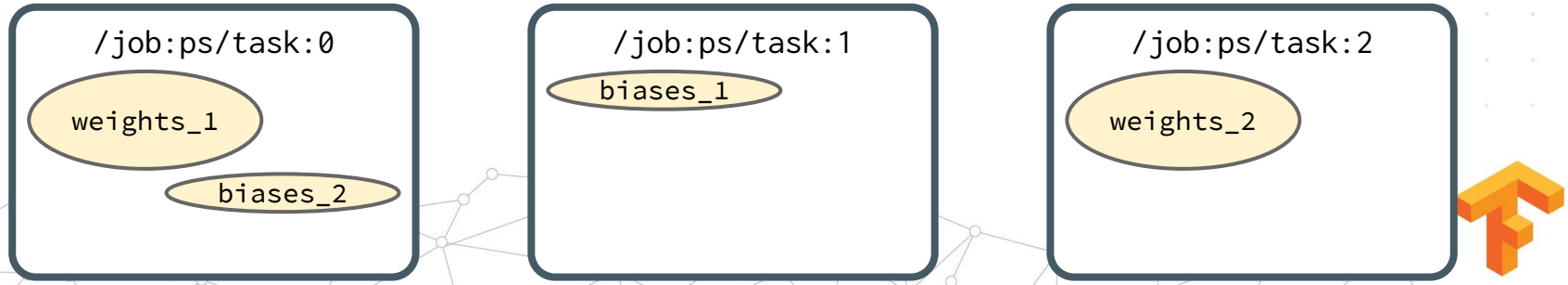
Round-robin variables

```
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])
```



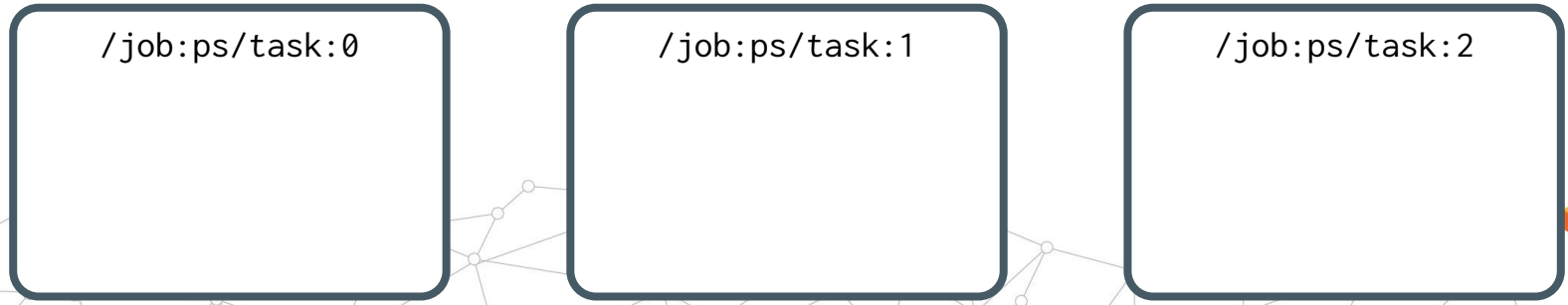
Round-robin variables

```
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])
```



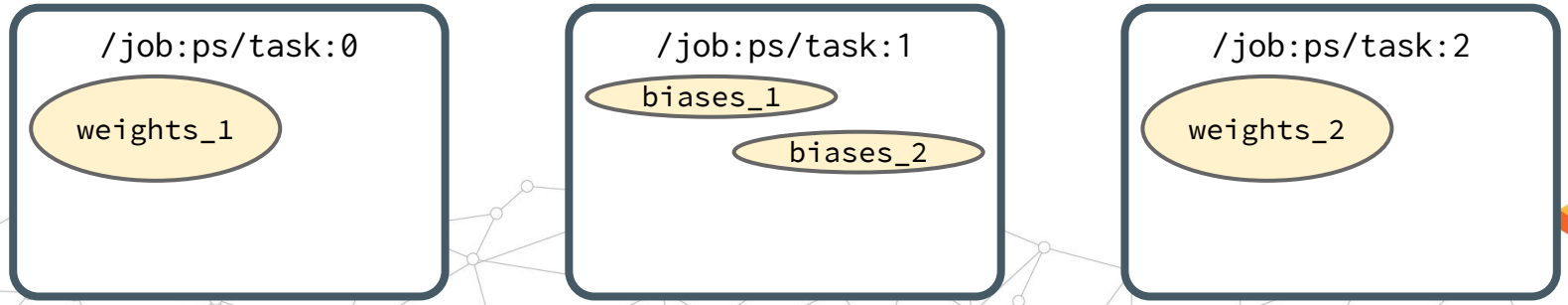
Load balancing variables

```
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])
```



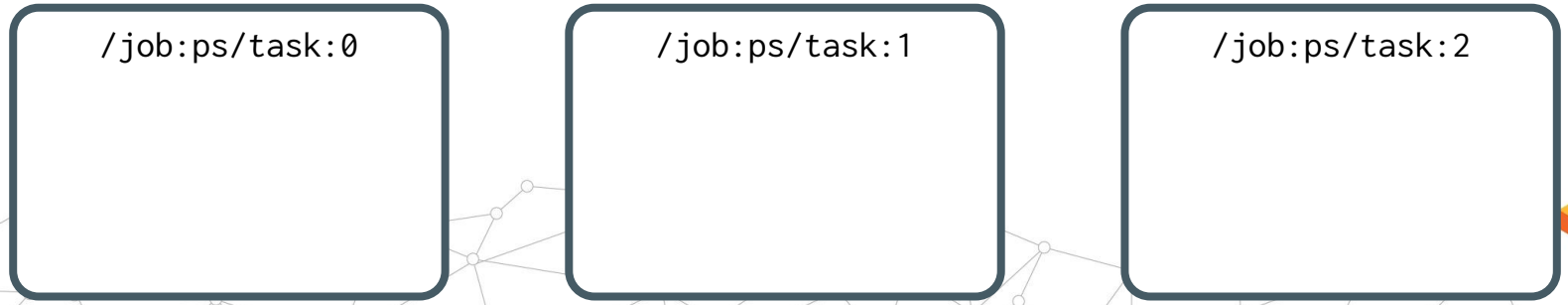
Load balancing variables

```
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])
```



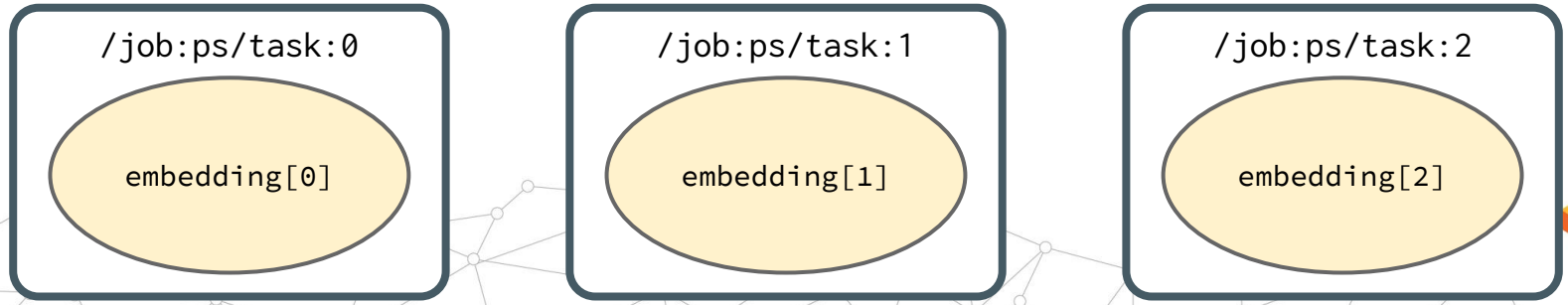
Partitioned variables

```
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))
```



Partitioned variables

```
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))
```

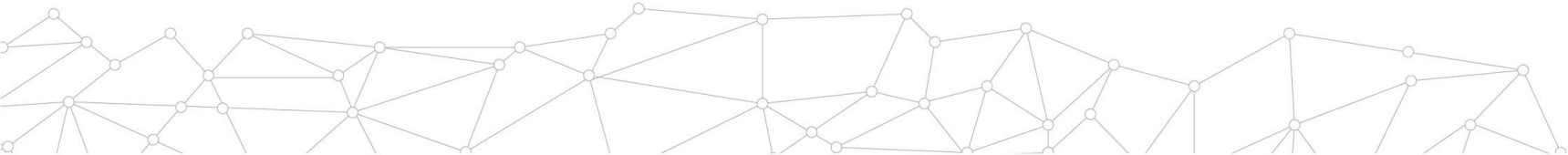


Device placement

`tf.train.replica_device_setter()`

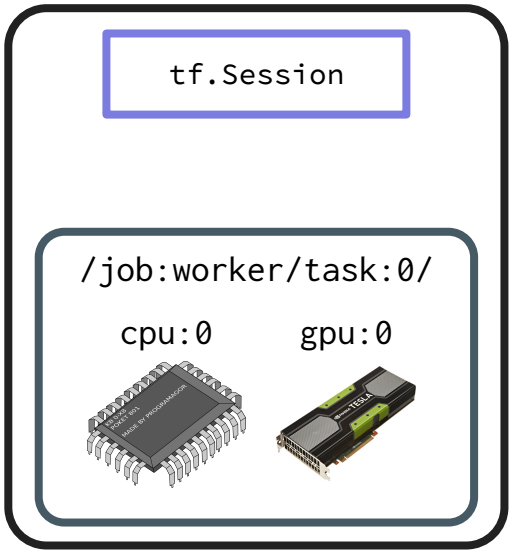
Simple heuristic for between-graph partitioning

- Round-robin variable placement by default
- Optional strategy for load balancing, partitioning
- All other ops placed on a worker task
- Customize using nested `with tf.device(…):` blocks



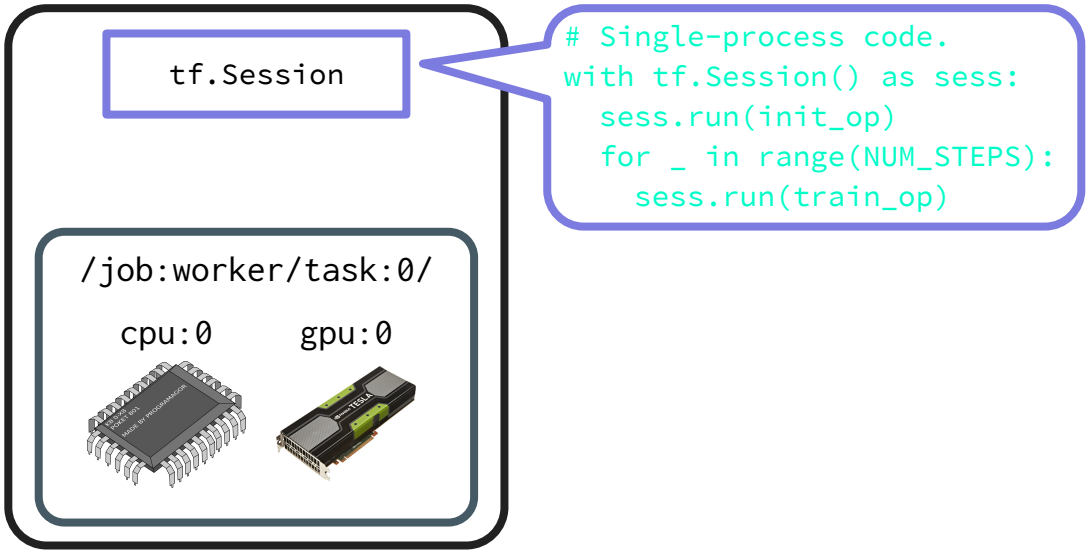
Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*



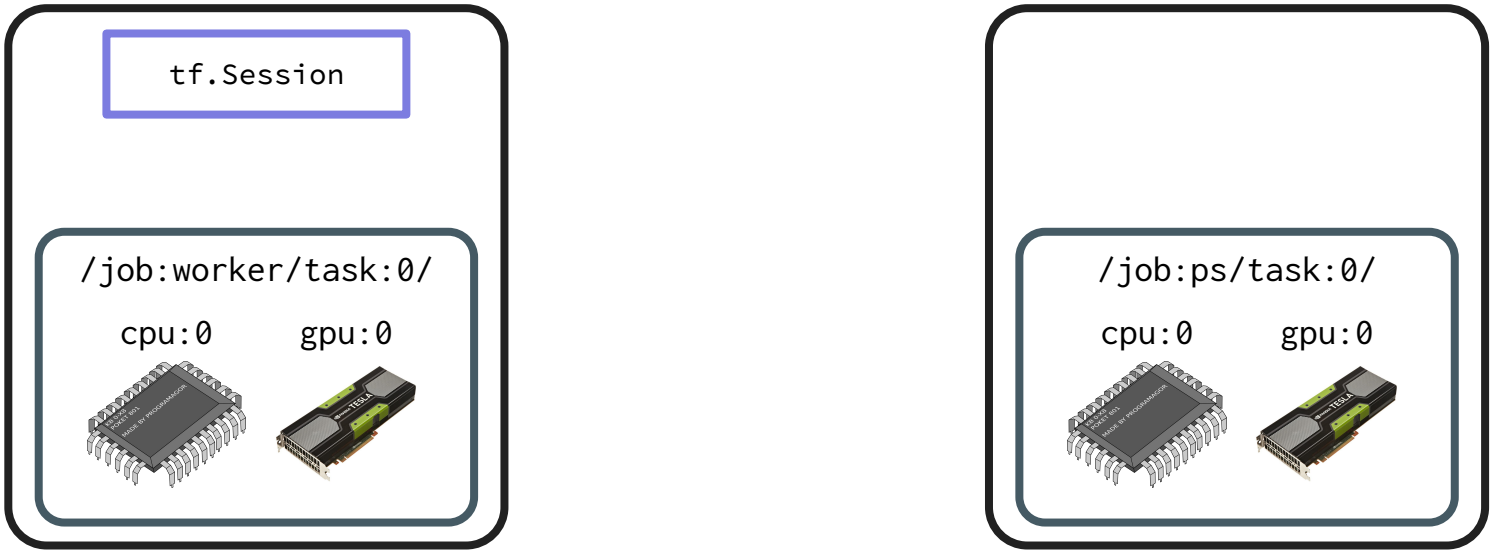
Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*



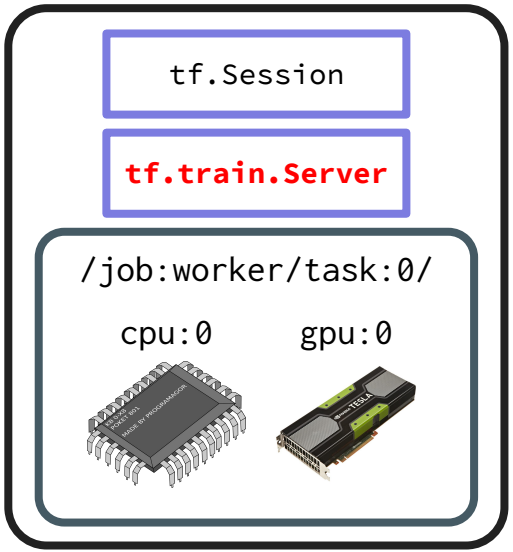
Sessions and Servers

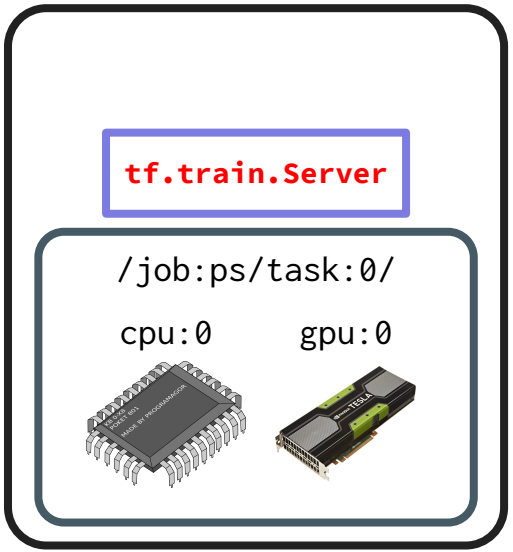
Distributed TensorFlow runs on a *cluster* of *servers*



Sessions and Servers

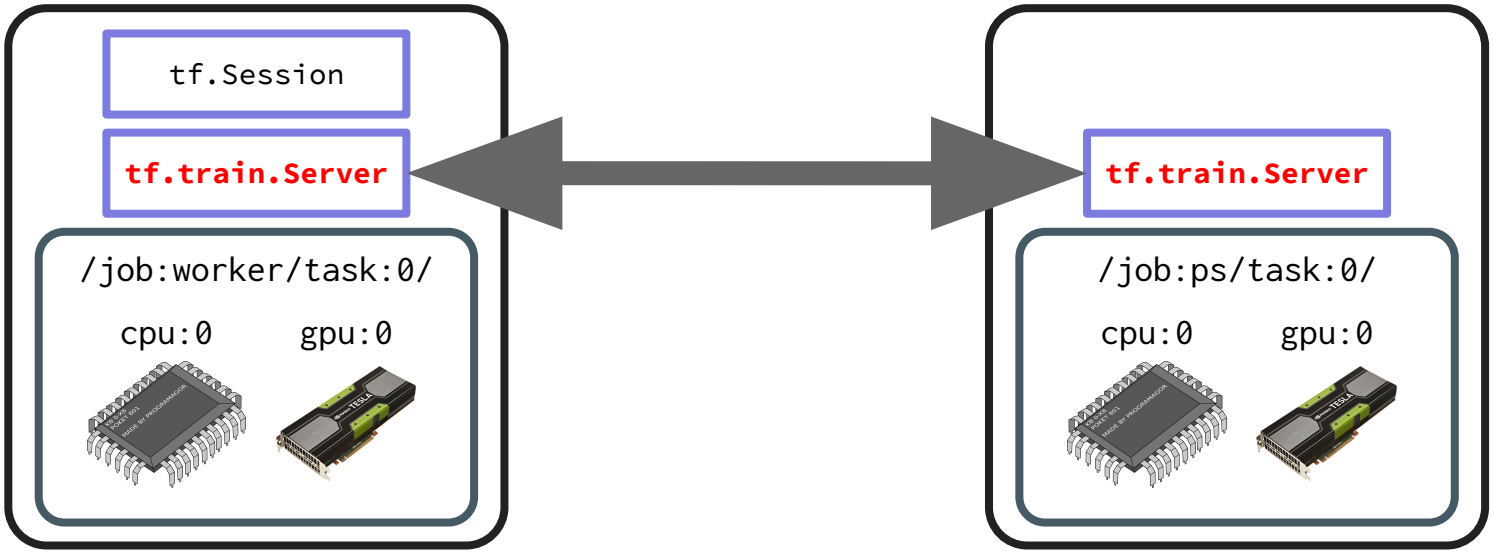
Distributed TensorFlow runs on a *cluster* of *servers*





Sessions and Servers

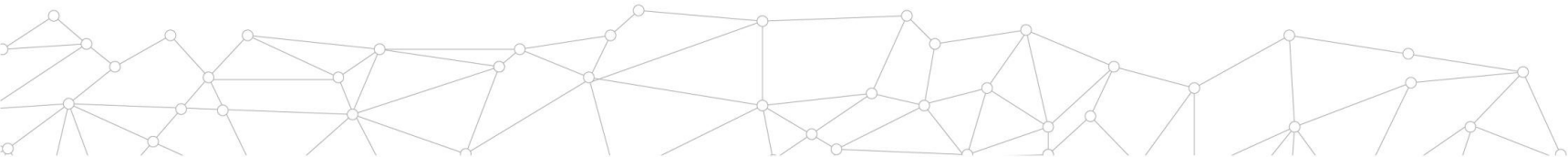
Distributed TensorFlow runs on a *cluster* of *servers*



Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

```
# 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:
  # …
```



Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

`cluster` defines the set of processes

```
# 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:
  # …
```

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

`cluster` defines the set of processes

```
# 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:
  # …
```

`server` implements a particular task in `cluster`

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

`cluster` defines the set of processes

```
# 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:
  # …
```

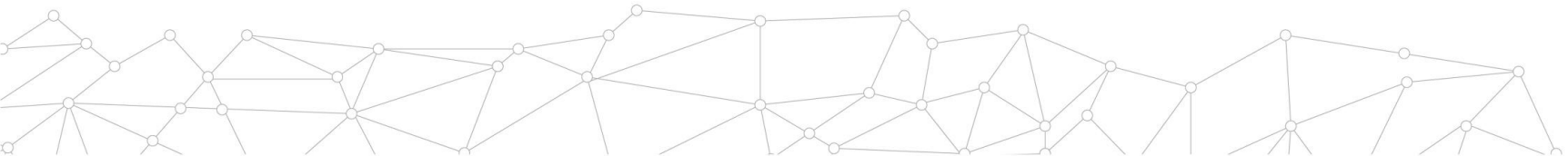
`server` implements a particular task in `cluster`

`sess` can run code on any device in `cluster`

Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

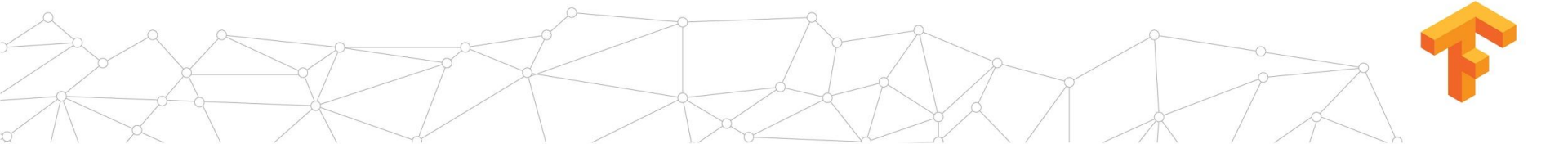
```
# Distributed code for a PS task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

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

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



Sessions and Servers

Distributed TensorFlow runs on a *cluster* of *servers*

```
# 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()
```



“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

Fault tolerance

```
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()

with tf.Session(server.target) as sess:
  while True:
    # …
    if step % 1000 == 0:
      saver.save(sess, "/home/mrry/…")
```

Fault tolerance

```
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, "/home/mrry/…")
```

Each PS task writes in parallel

Fault tolerance

```
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 is_chief and step % 1000 == 0:
      saver.save(sess, "/home/mrry/…")
```

Each PS task writes in parallel

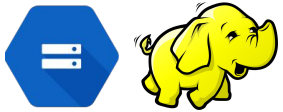
One worker task acts as "chief"

Fault tolerance

```
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 is_chief and step % 1000 == 0:
      saver.save(sess, "gs://mrry/model/…")
```

Each PS task writes in parallel

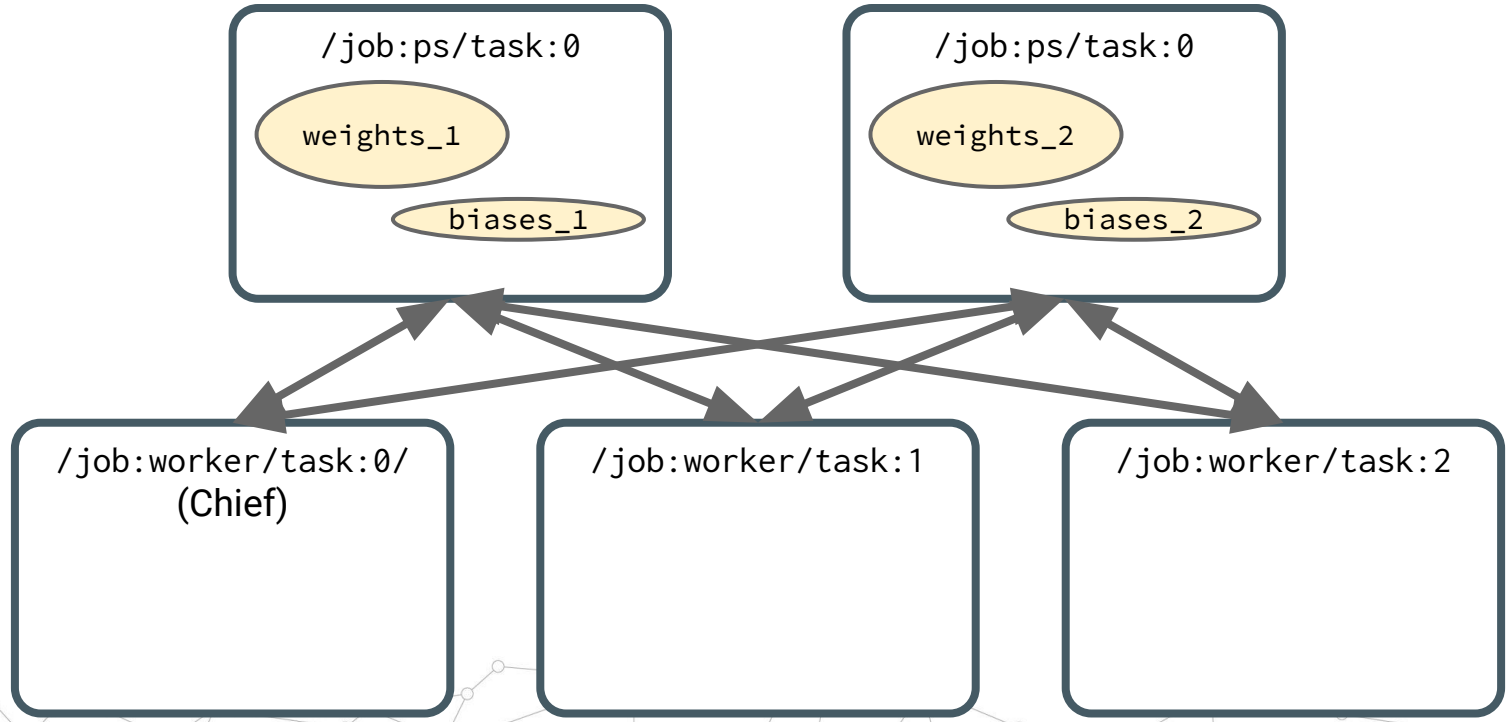
Saver can write checkpoints to a distributed file system



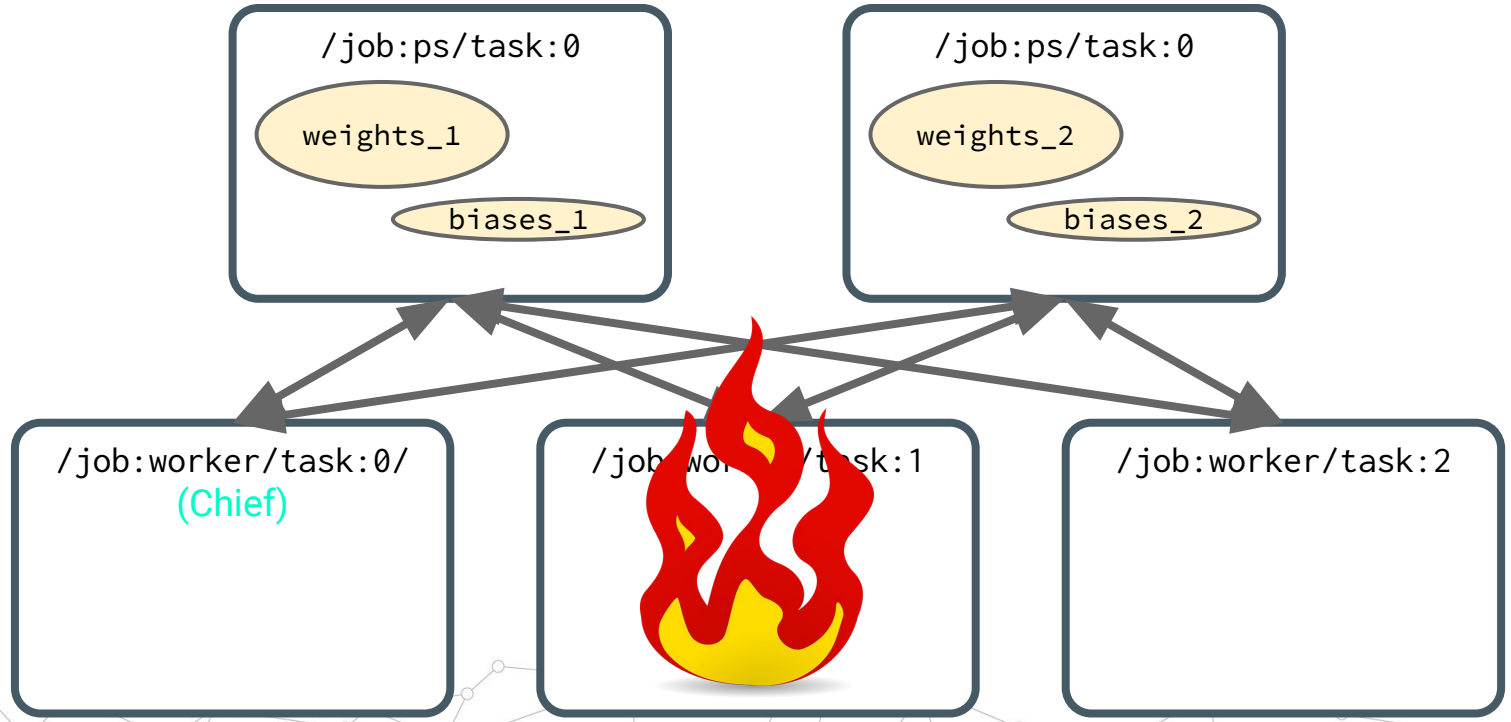
...

One worker task acts as "chief"

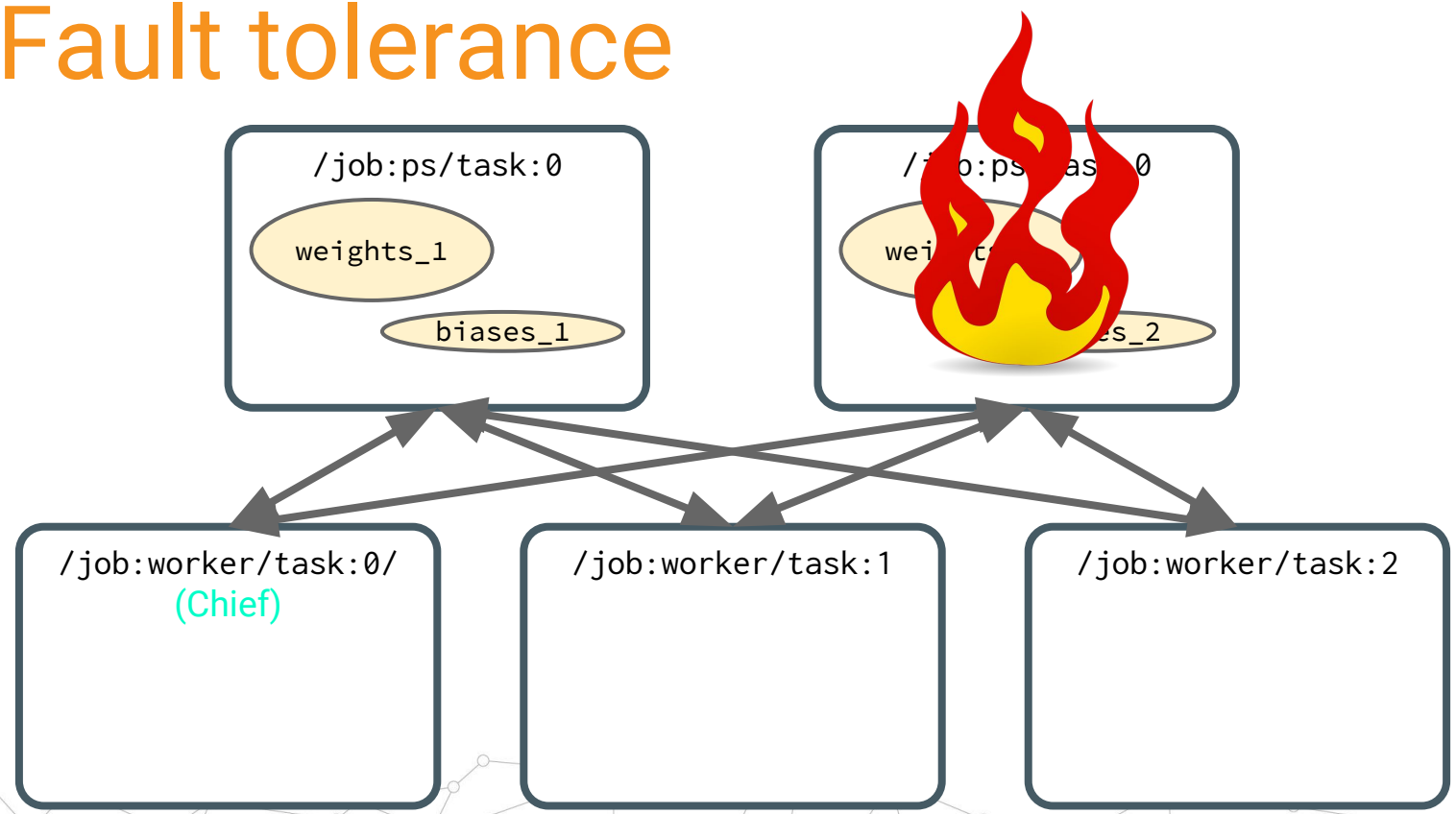
Fault tolerance



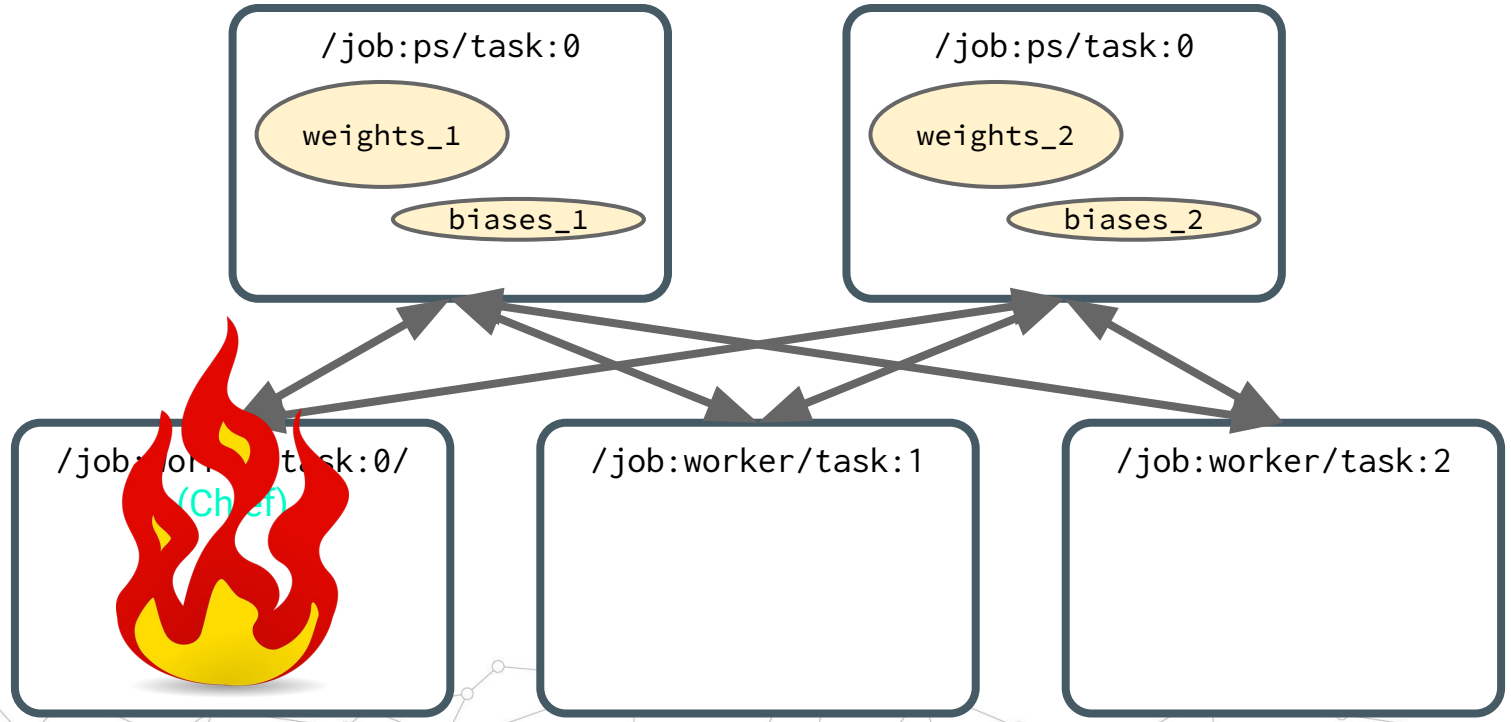
Fault tolerance



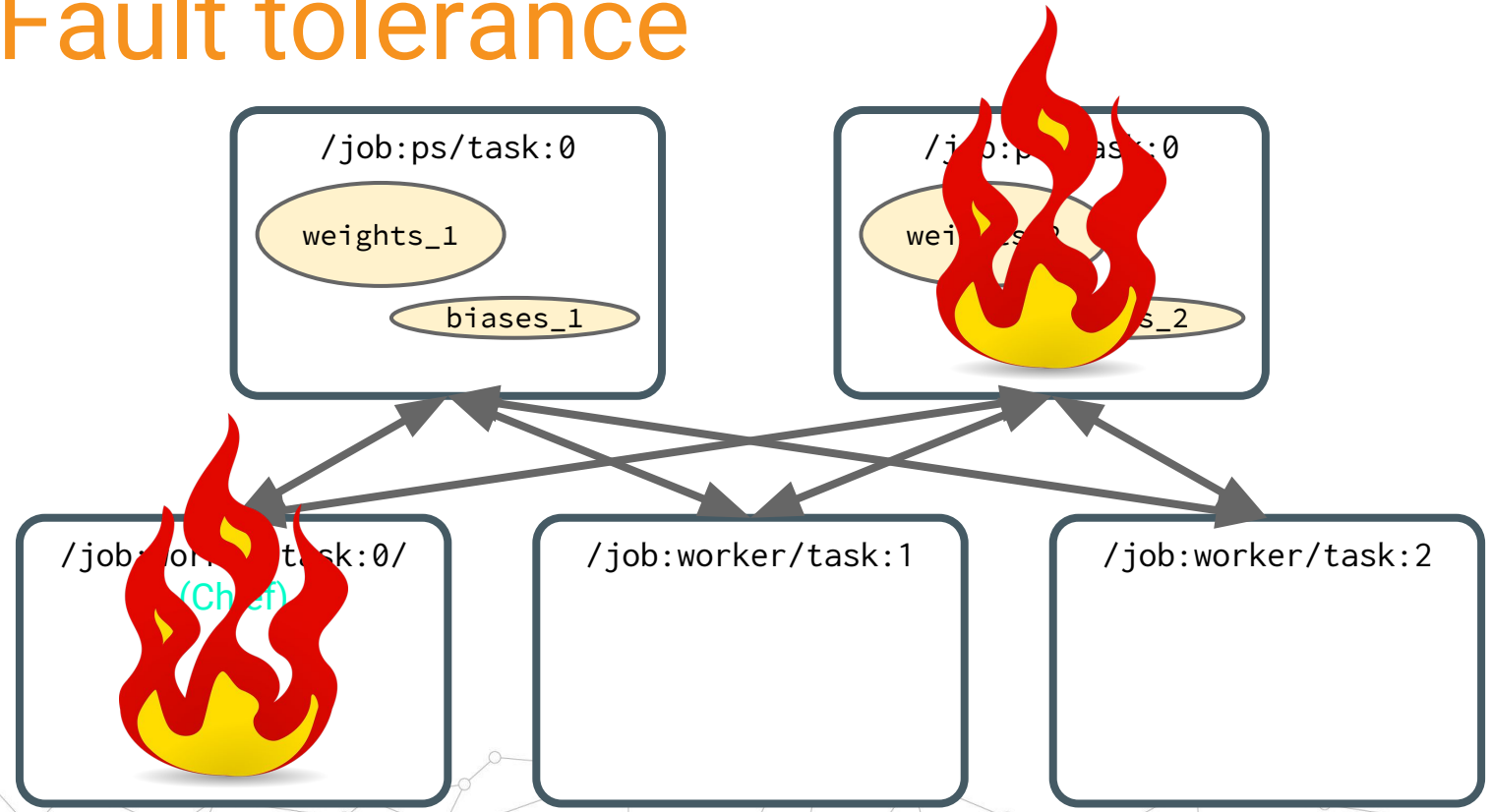
Fault tolerance



Fault tolerance



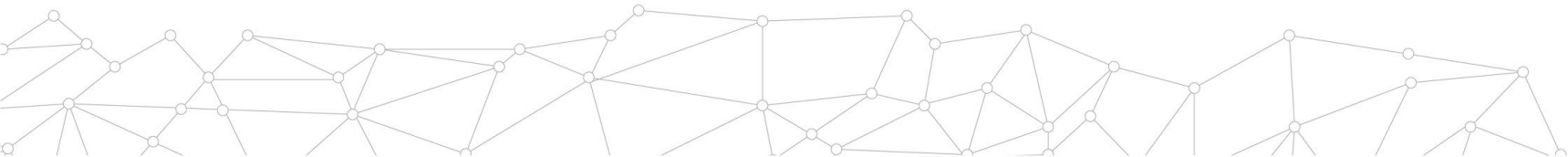
Fault tolerance



Fault tolerance

`MonitoredTrainingSession` automates the recovery process

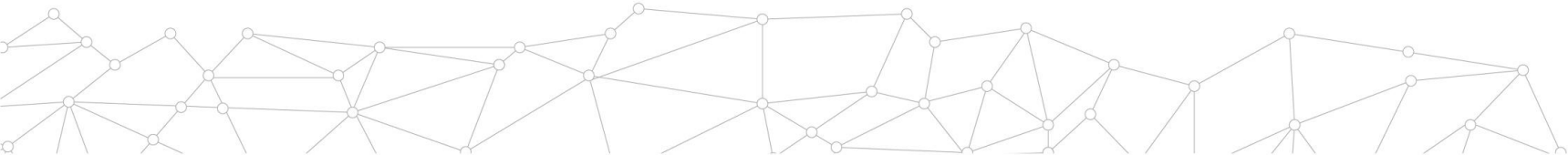
```
# Single-process code.
with tf.Session() as sess:
  sess.run(init_op)  # Or saver.restore(sess, …)
  for _ in range(NUM_STEPS):
    sess.run(train_op)
```



Fault tolerance

`MonitoredTrainingSession` automates the recovery process

```
# 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)
```



Fault tolerance

`MonitoredTrainingSession` automates the recovery process

```
# 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)
```

Automatically initializes and/or restores variables before returning

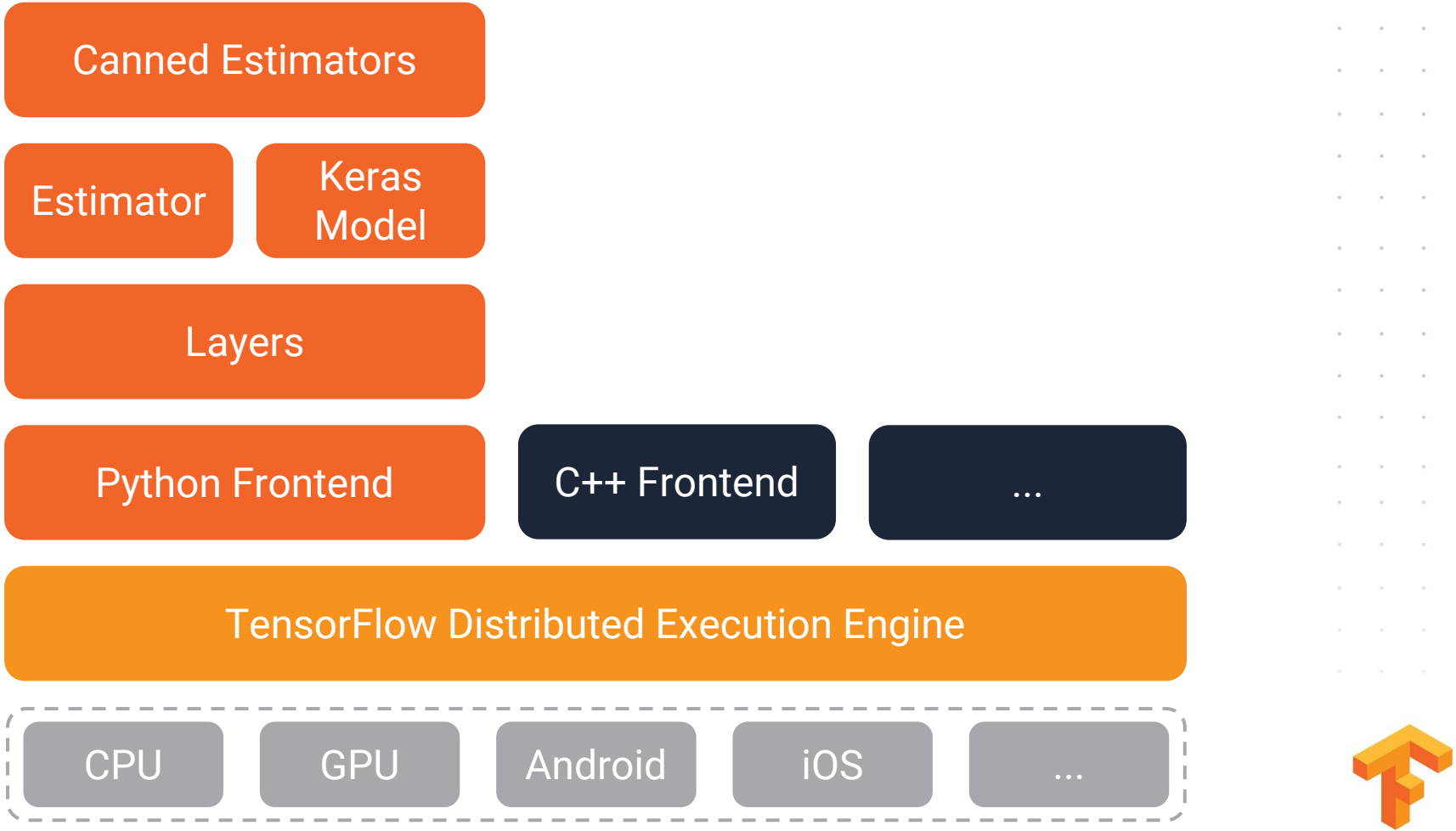
Fault tolerance

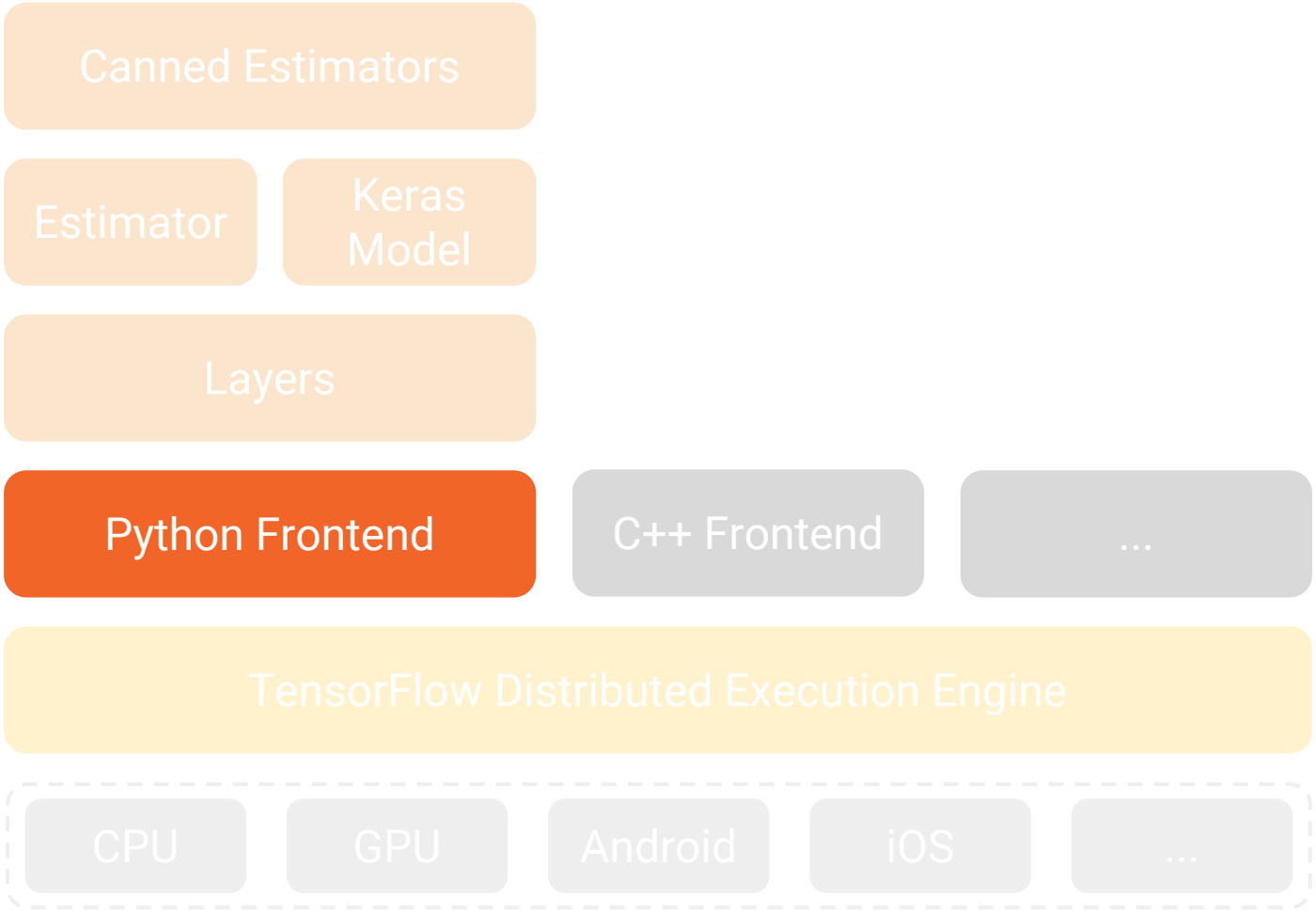
`MonitoredTrainingSession` automates the recovery process

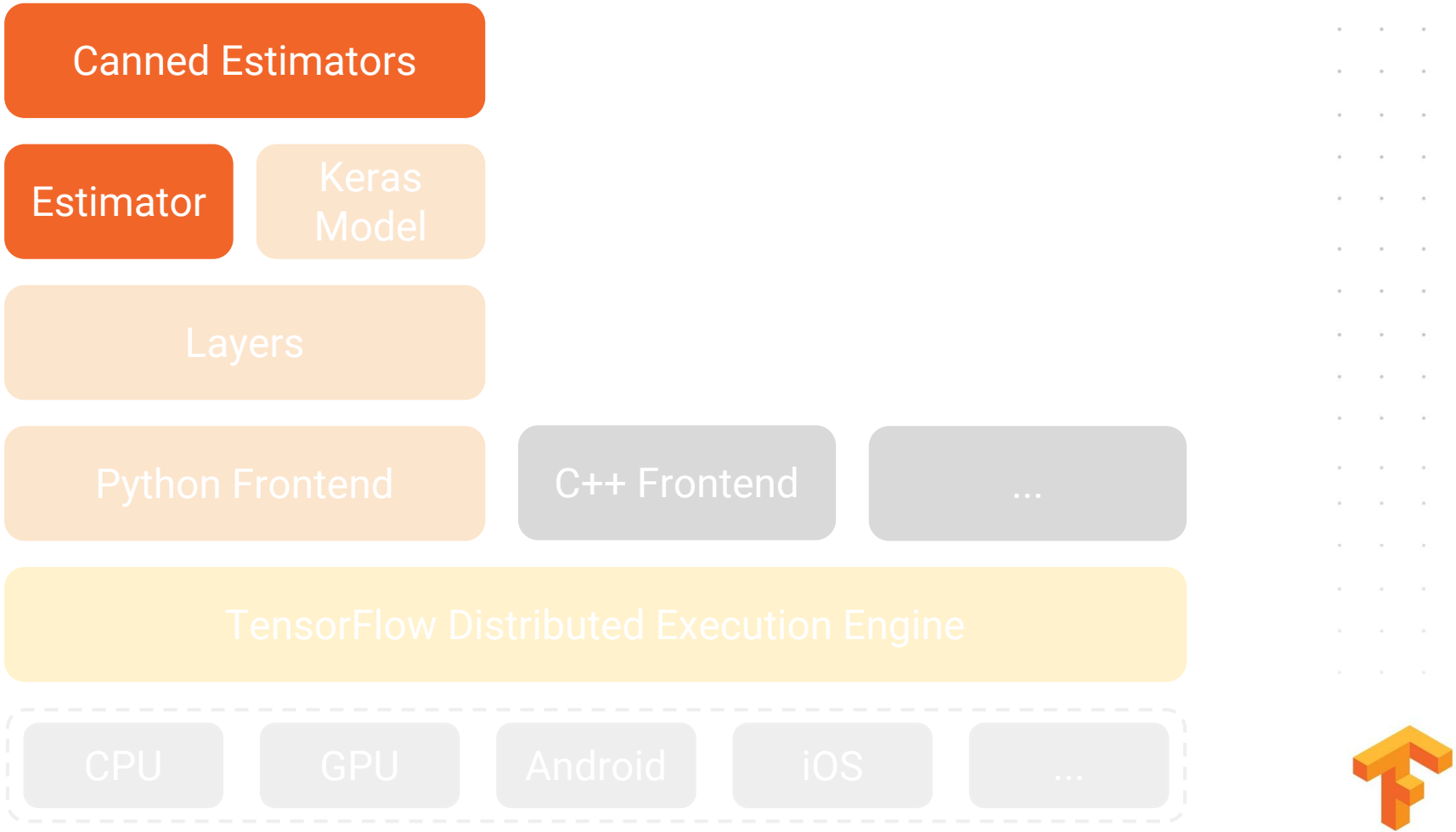
```
# Distributed code.
server = tf.train.Server(…)
is_chief = FLAGS.task_index == 0
with tf.train.MonitoredTrainingSession(server.target, is_chief) as sess:
  while not sess.should_stop():
    sess.run(train_op)
```

Automatically initializes and/or restores variables before returning

`MonitoredSession.run()` automatically recovers from PS failures, and can run additional code in *hooks*


Canned Estimators
Estimator
Keras Model
Layers
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...


Canned Estimators
Estimator
Keras Model
Layers
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...


Canned Estimators
Estimator
Keras Model
Layers
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...

Experiments and Estimators

High-level APIs package up the whole distributed workflow

```
def experiment_fn(config, params):
  features = [tf.layers.embedding_column(…),
              tf.layers.bucketized_column(…)]
  return Experiment(
      train_input_fn=…, eval_input_fn=…,
      estimator=DNNClassifier(
          hidden_units=[10, 20], feature_columns=features,
          config, params))

config = tf.train.parse_config_from_env()
experiment.run(experiment_fn, config, …)
```

Experiments and Estimators

High-level APIs package up the whole distributed workflow

```
def experiment_fn(config, params):
  features = [tf.layers.embedding_column(…),
              tf.layers.bucketized_column(…)]
  return Experiment(
      train_input_fn=…, eval_input_fn=…,
      estimator=DNNClassifier(
          hidden_units=[10, 20], feature_columns=features,
          config, params))

config = tf.train.parse_config_from_env()
experiment.run(experiment_fn, config, …)
```

Declarative specification of a fully-connected neural network

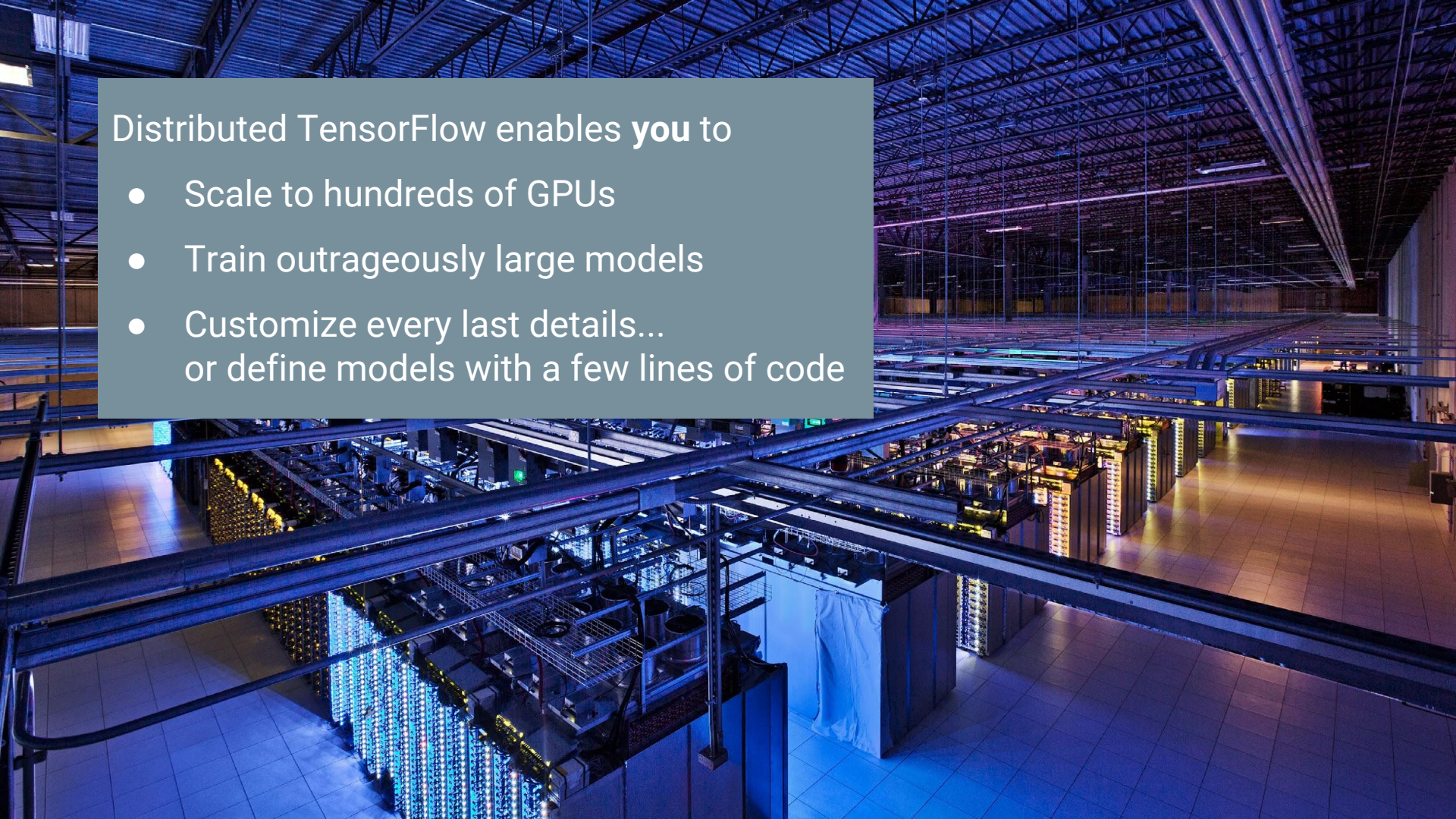
Experiments and Estimators

High-level APIs package up the whole distributed workflow

```
def experiment_fn(config, params):
  features = [tf.layers.embedding_column(…),
              tf.layers.bucketized_column(…)]
  return Experiment(
      train_input_fn=…, eval_input_fn=…,
      estimator=DNNClassifier(
          hidden_units=[10, 20], feature_columns=features,
          config, params))

config = tf.train.parse_config_from_env()
experiment.run(experiment_fn, config, …)
```

Declarative specification of a fully-connected neural network

Runs training, evaluation, etc. on Cloud ML


Distributed TensorFlow enables **you** to
Scale to hundreds of GPUs
Train outrageously large models
Customize every last details...
or define models with a few lines of code

Distributed TensorFlow enables **you** to

- Scale to hundreds of GPUs
- Train outrageously large models
- Customize every last details...
  or define models with a few lines of code

Further reading:

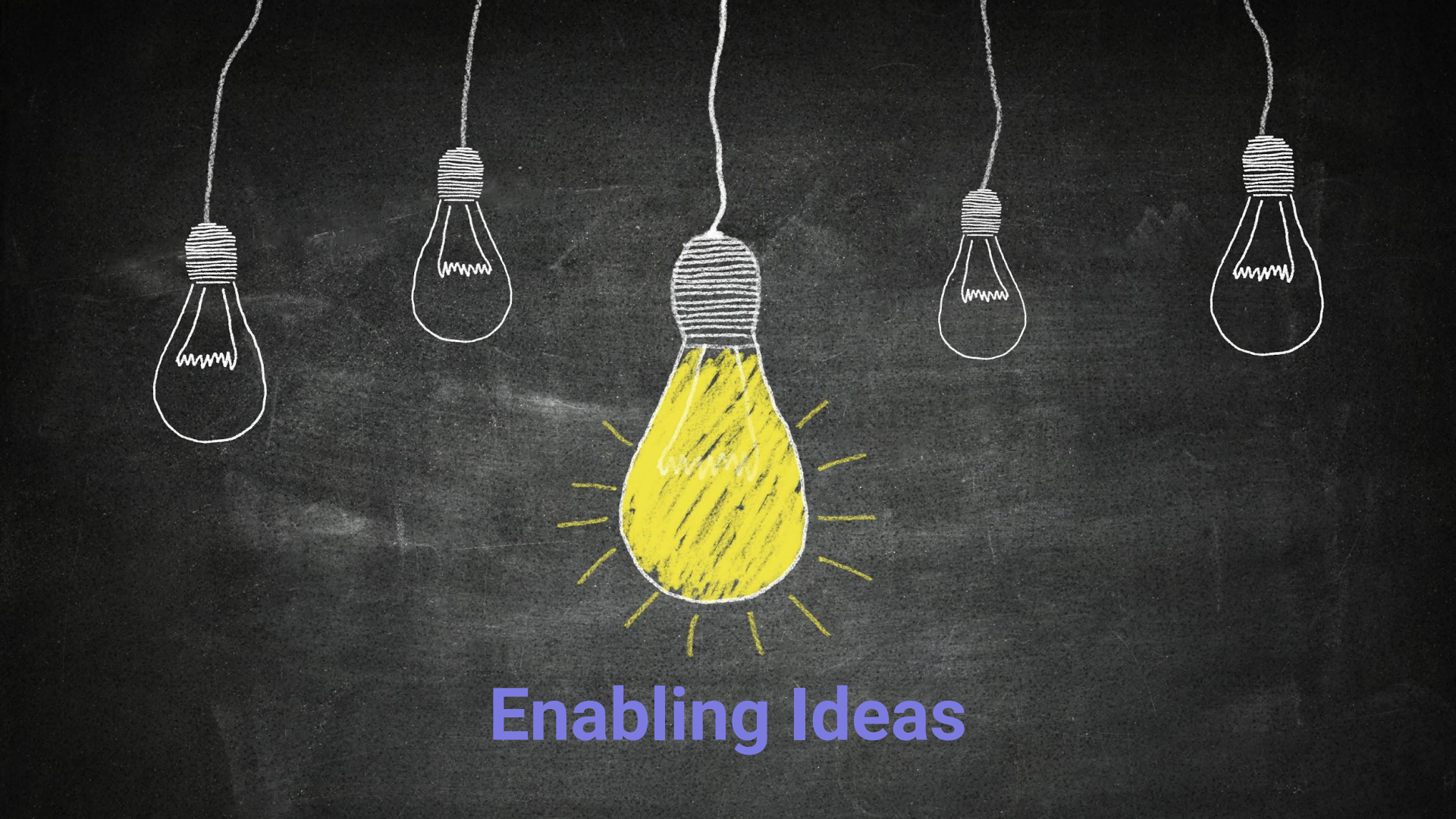
`tensorflow.org/architecture`

`tensorflow.org/how_tos/distributed`

`tensorflow.org/tutorials/estimators`


OP 1
OP 2
A


GOALS


Enabling Ideas

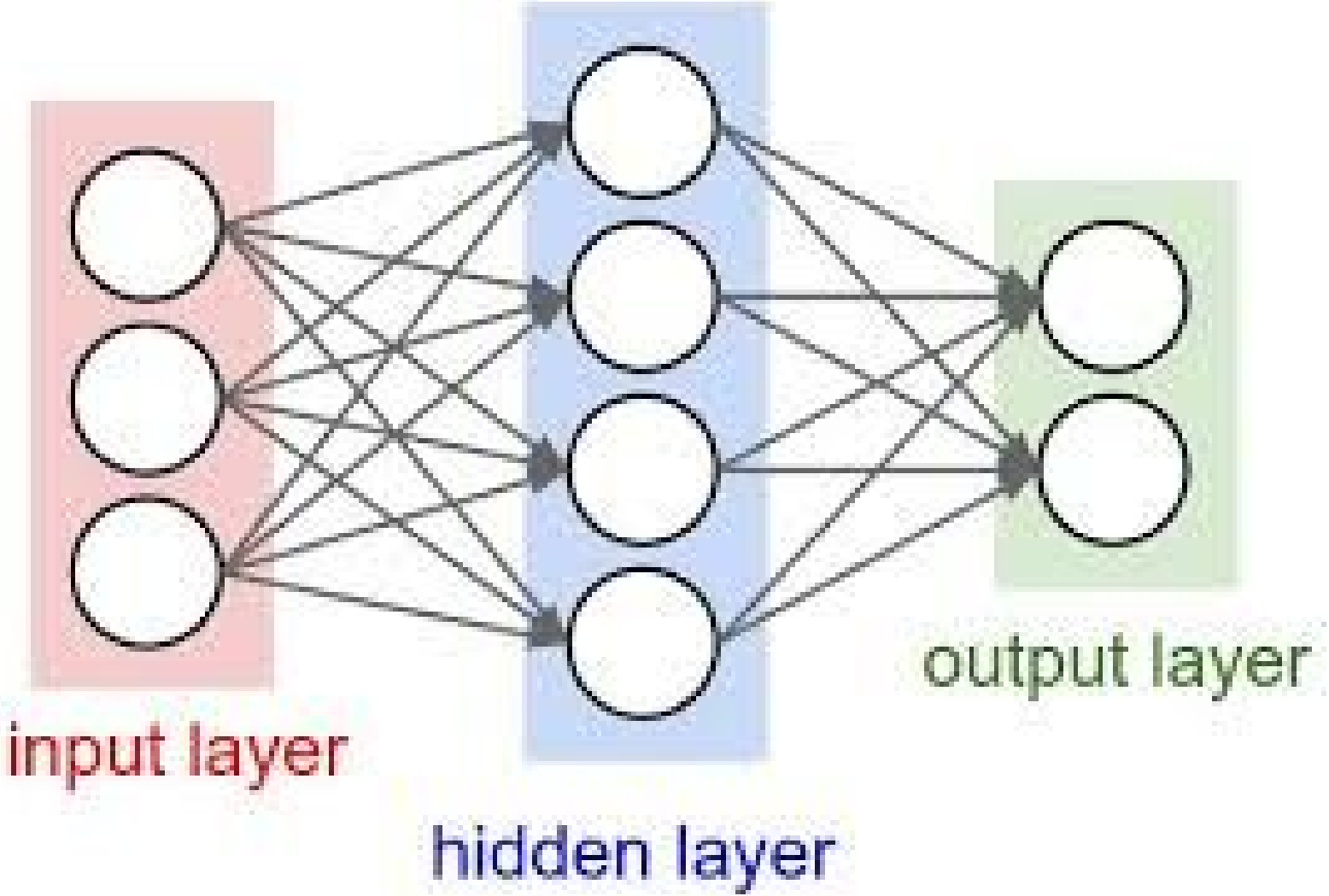

Smarter Products

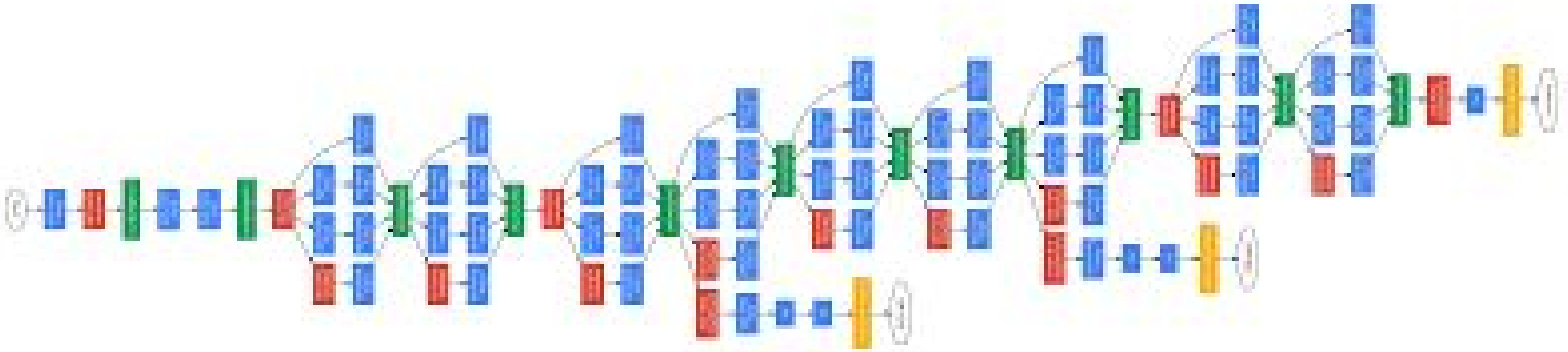
Accelerating Innovation

Scaling Impact

Companies Using TensorFlow


COMMUNITY


input layer
hidden layer
output layer



Open Source Models: from Google and *Many* Others

Fast Style Transfer https://github.com/lengstrom/fast-style-transfer/

Colornet https://github.com/pavelgonchar/colornet

Super Resolution https://github.com/david-gpu/srez

TTS https://github.com/ibab/tensorflow-wavenet

Speech Recongition https://github.com/buriburisuri/speech-to-text-wavenet

Many more https://github.com/jtoy/awesome-tensorflow

Flexibility is key

- Experiment support for Imperative-style TF
- Provide constructs needed to support research and production use cases
  - Conditionals/Loops

Wide Variety of Research using TF

Music
https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning/

WaveNet https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Translation
https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

Summarization
https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html

Show and Tell
https://research.googleblog.com/2016/09/show-and-tell-image-captioning-open.html

Inception

What's next in V 1.0

New Features

- TensorFlow Debugger (tfdbg)
- Java API (Experimental)
- Android Demos: Person Detection / Camera-based image stylization

Stability

- Backwards Compatibility

Higher level APIs

tf.contrib.learn

https://www.tensorflow.org/versions/r0.12/tutorials/tflearn/index.html

-

tf.slim

https://research.googleblog.com/2016/08/tf-slim-high-level-library-to-define.html

Keras

https://keras.io/

High-level questions?