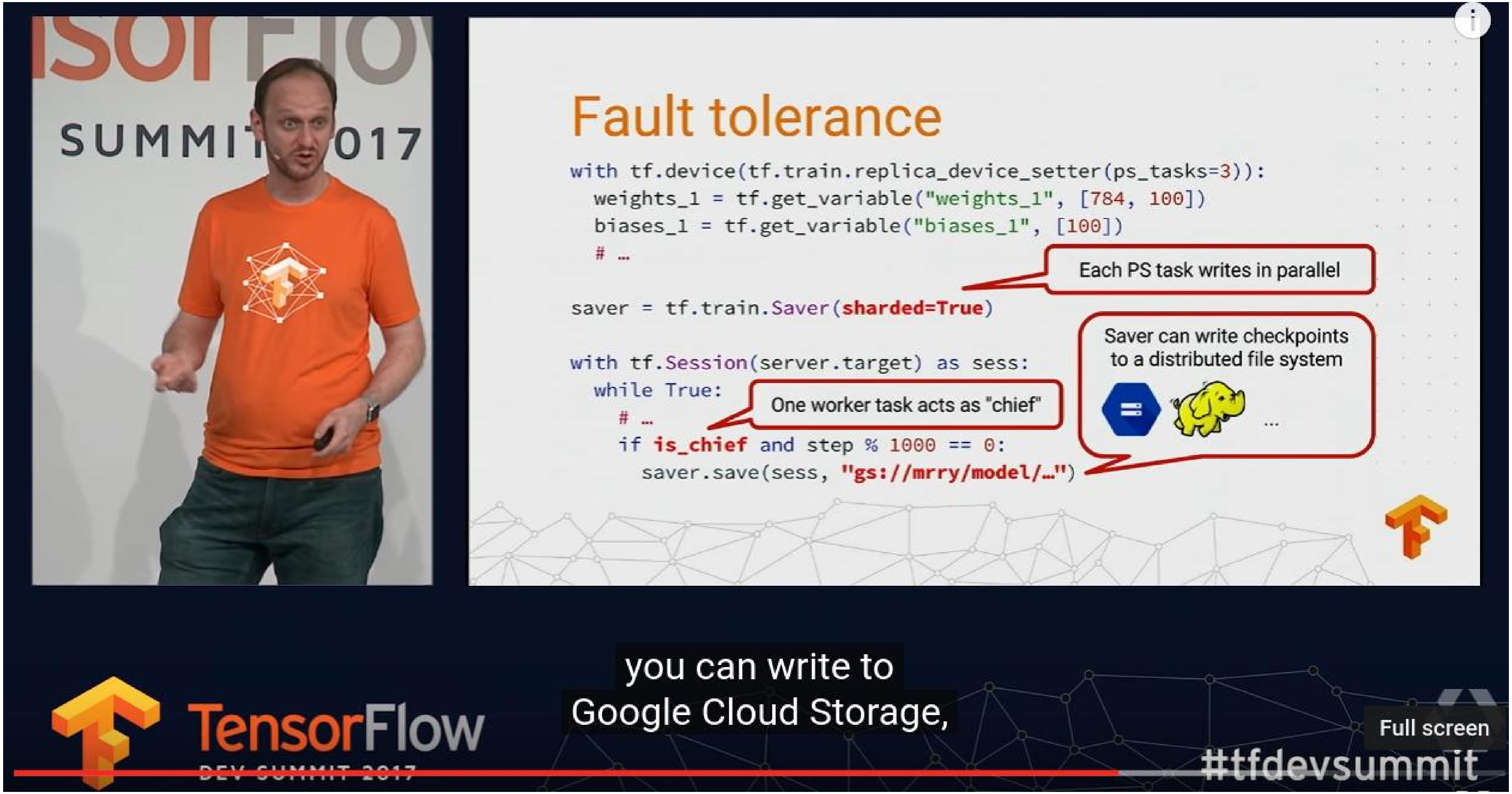
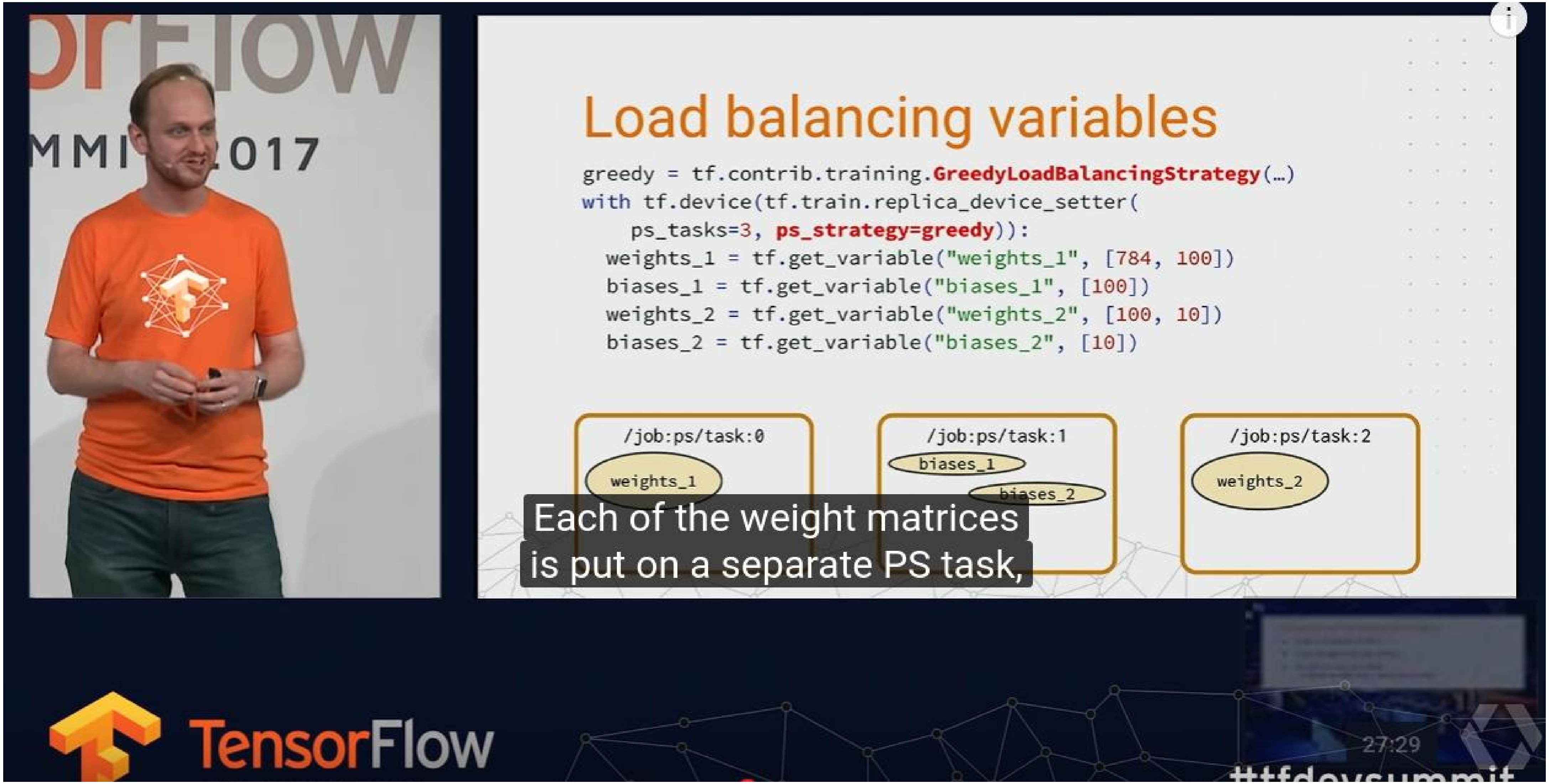
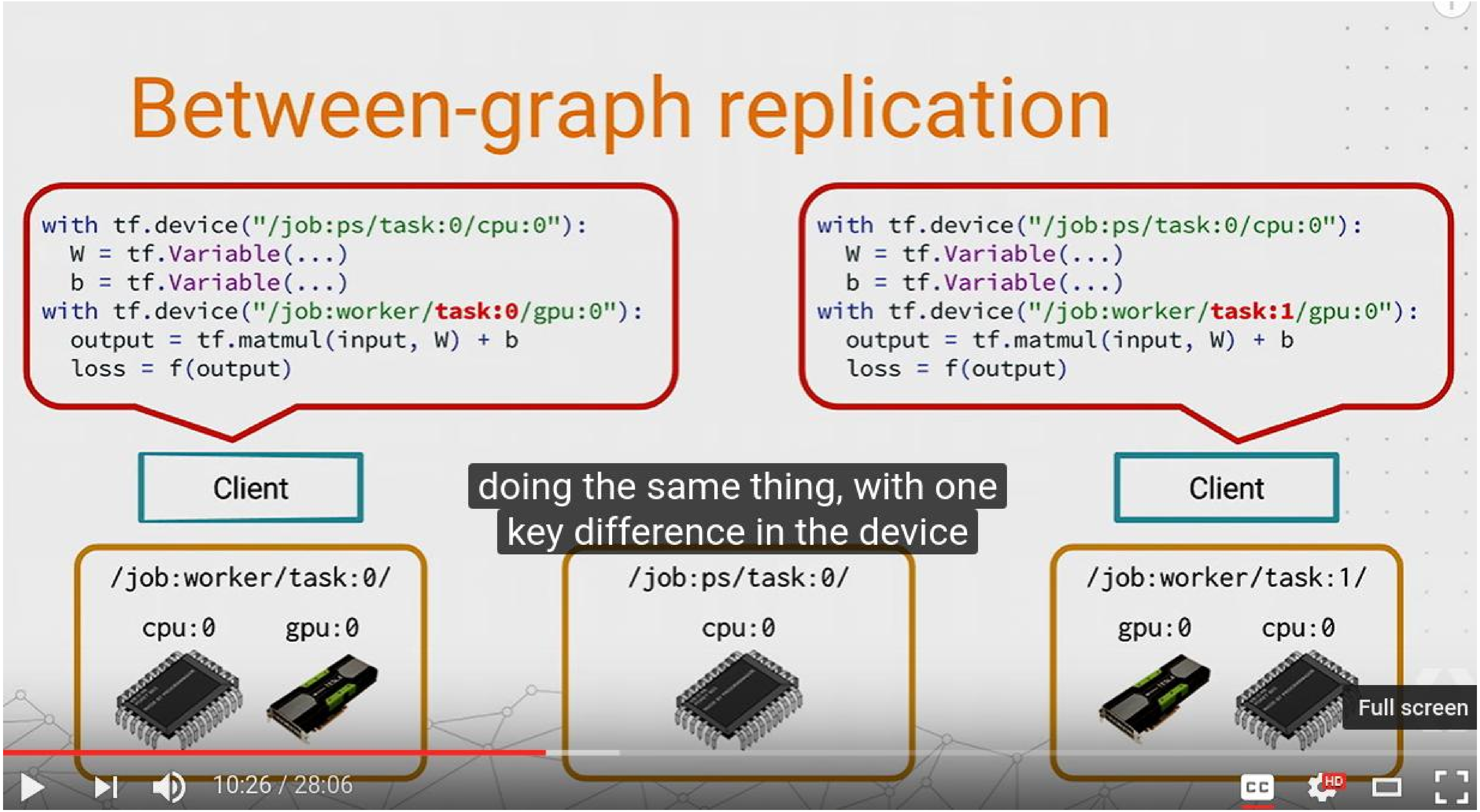
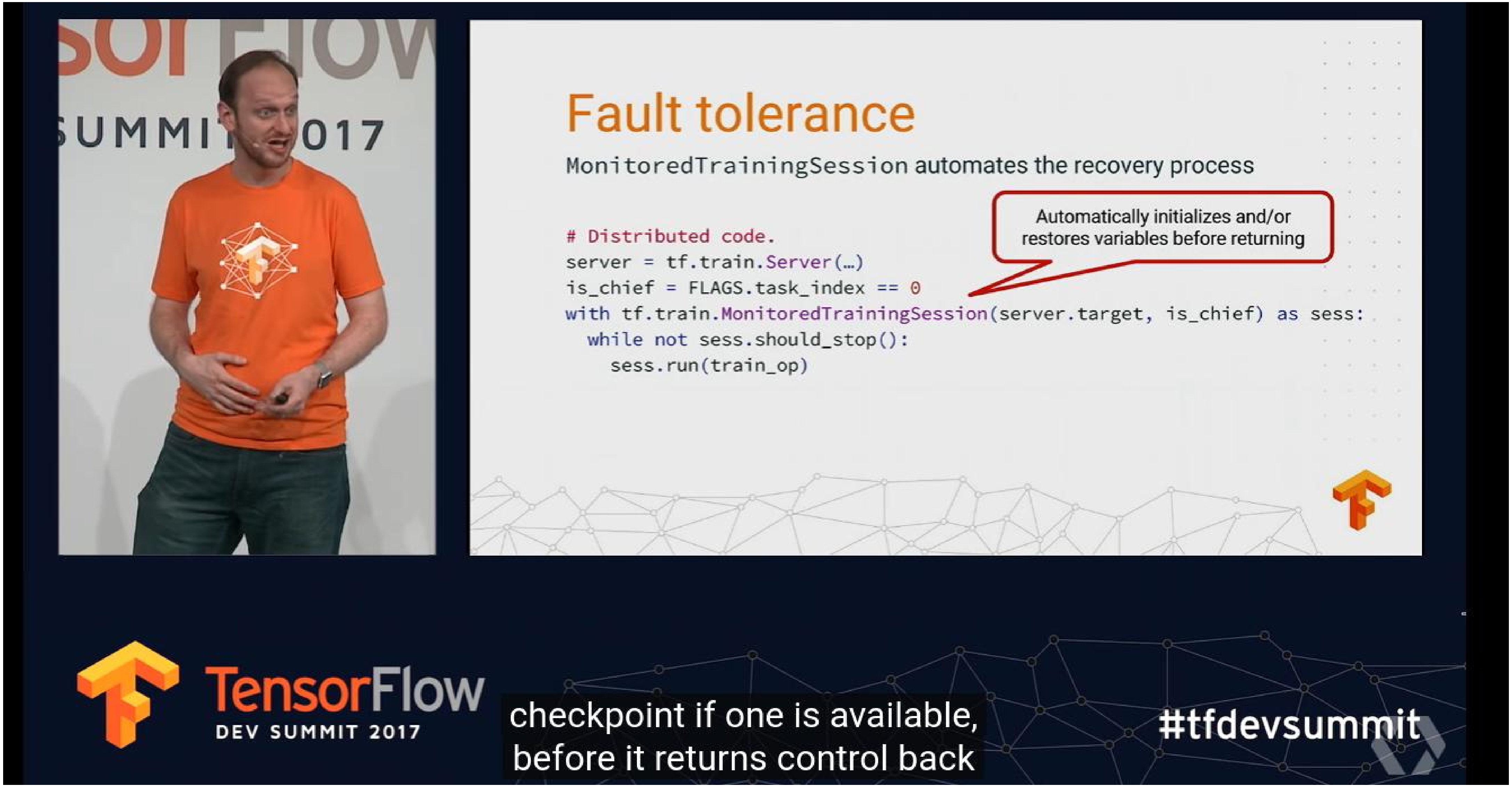
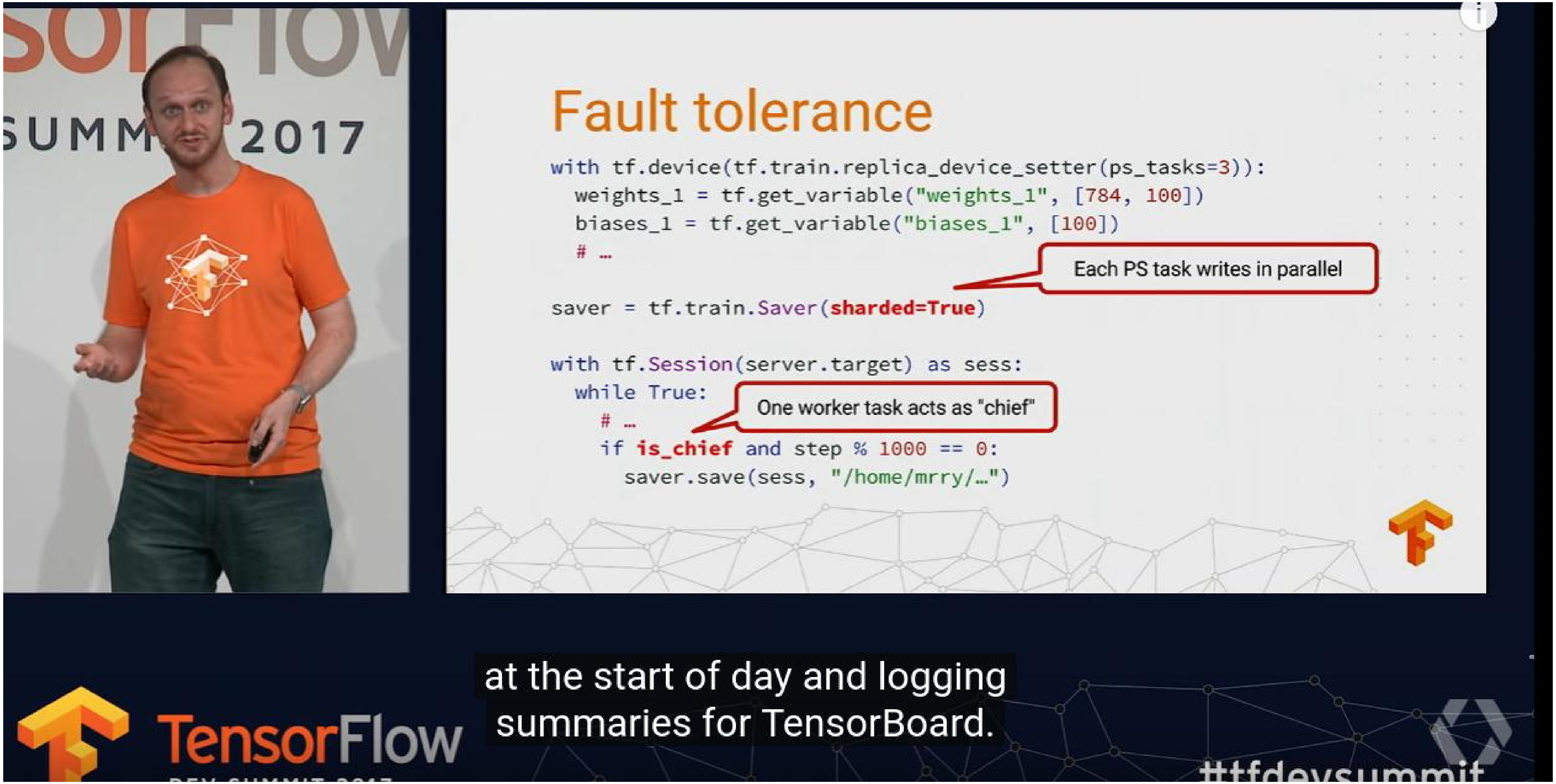

SUMMIT 017
Fault tolerance
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):
  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  # …
saver = tf.train.Saver(sharded=True)
with tf.Session(server.target) as sess:
  while True:
    # …
    if 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"
...
you can write to
Google Cloud Storage,
TensorFlow
DEV SUMMIT 2017
Full screen
#tfdevsummit

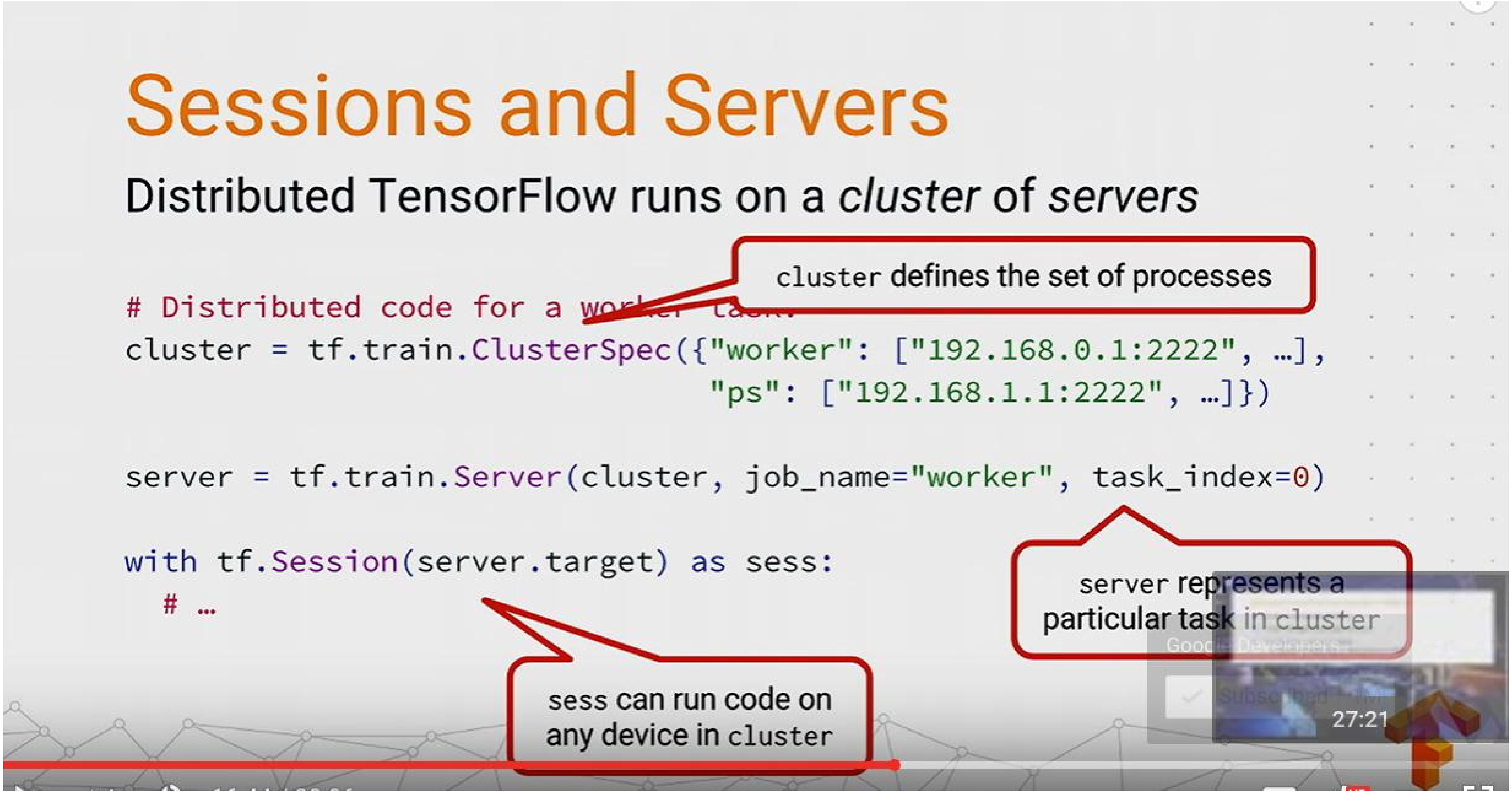

Load balancing variables
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(…)
with tf.device(tf.train.replica_device_setter(
ps_tasks=3, ps_strategy=greedy)):
weights_1 = tf.get_variable("weights_1", [784, 100])
biases_1 = tf.get_variable("biases_1", [100])
weights_2 = tf.get_variable("weights_2", [100, 10])
biases_2 = tf.get_variable("biases_2", [10])
/job:ps/task:0
weights_1
/job:ps/task:1
biases_1
biases_2
/job:ps/task:2
weights_2
Each of the weight matrices
is put on a separate PS task,
TensorFlow
27:29

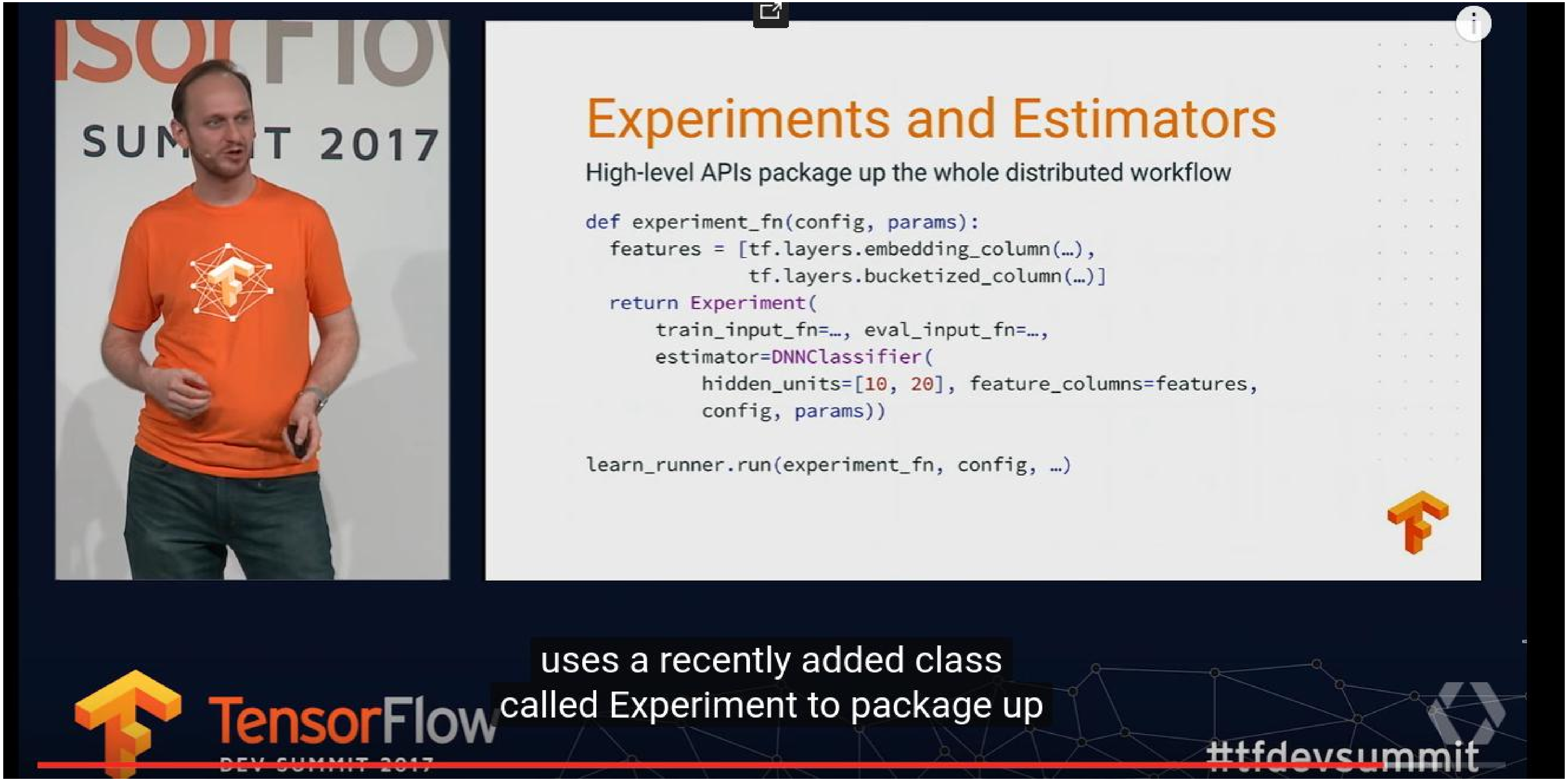

Between-graph replication
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
with tf.device("/job:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
  output = tf.matmul(input, W) + b
  loss = f(output)
Client
Client
doing the same thing, with one
key difference in the device
/job:worker/task:0/
cpu:0
gpu:0
/job:ps/task:0/
cpu:0
/job:worker/task:1/
gpu:0
cpu:0
Full screen
10:26 / 28:06


SUMMIT 2017
Fault tolerance
MonitoredTrainingSession automates the recovery process
Automatically initializes and/or restores variables before returning
# 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)
TensorFlow
DEV SUMMIT 2017
checkpoint if one is available,
before it returns control back
#tfdevsummit



SUMM 2017
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])
# …
Each PS task writes in parallel
saver = tf.train.Saver(sharded=True)
with tf.Session(server.target) as sess:
while True:
One worker task acts as "chief"
# …
if is_chief and step % 1000 == 0:
saver.save(sess, "/home/mrry/…")
at the start of day and logging
summaries for TensorBoard.
TensorFlow

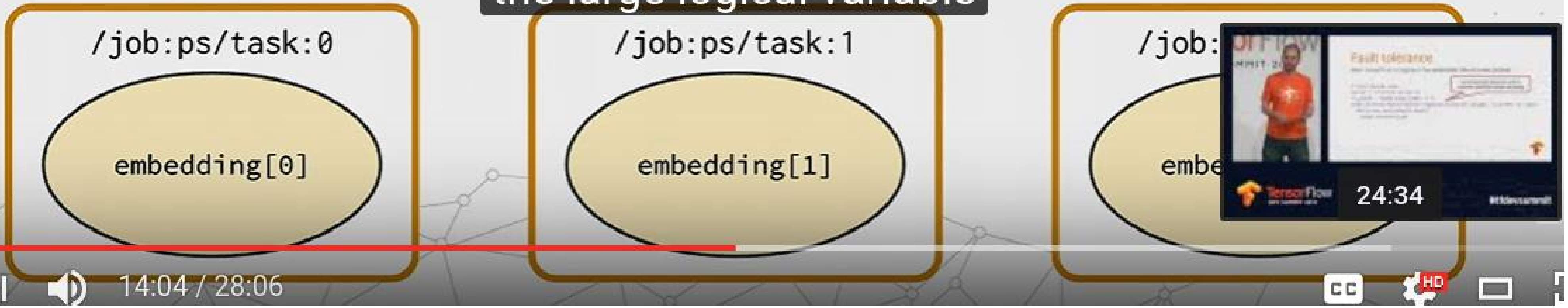

Sessions and Servers
Distributed TensorFlow runs on a cluster of servers
cluster defines the set of processes
# Distributed code for a
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
"ps": ["192.168.1.1:2222", …]})
server = tf.train.Server(cluster, job_name="worker", task_index=0)
server represents a
particular task in cluster
with tf.Session(server.target) as sess:
…
sess can run code on
any device in cluster
27:21

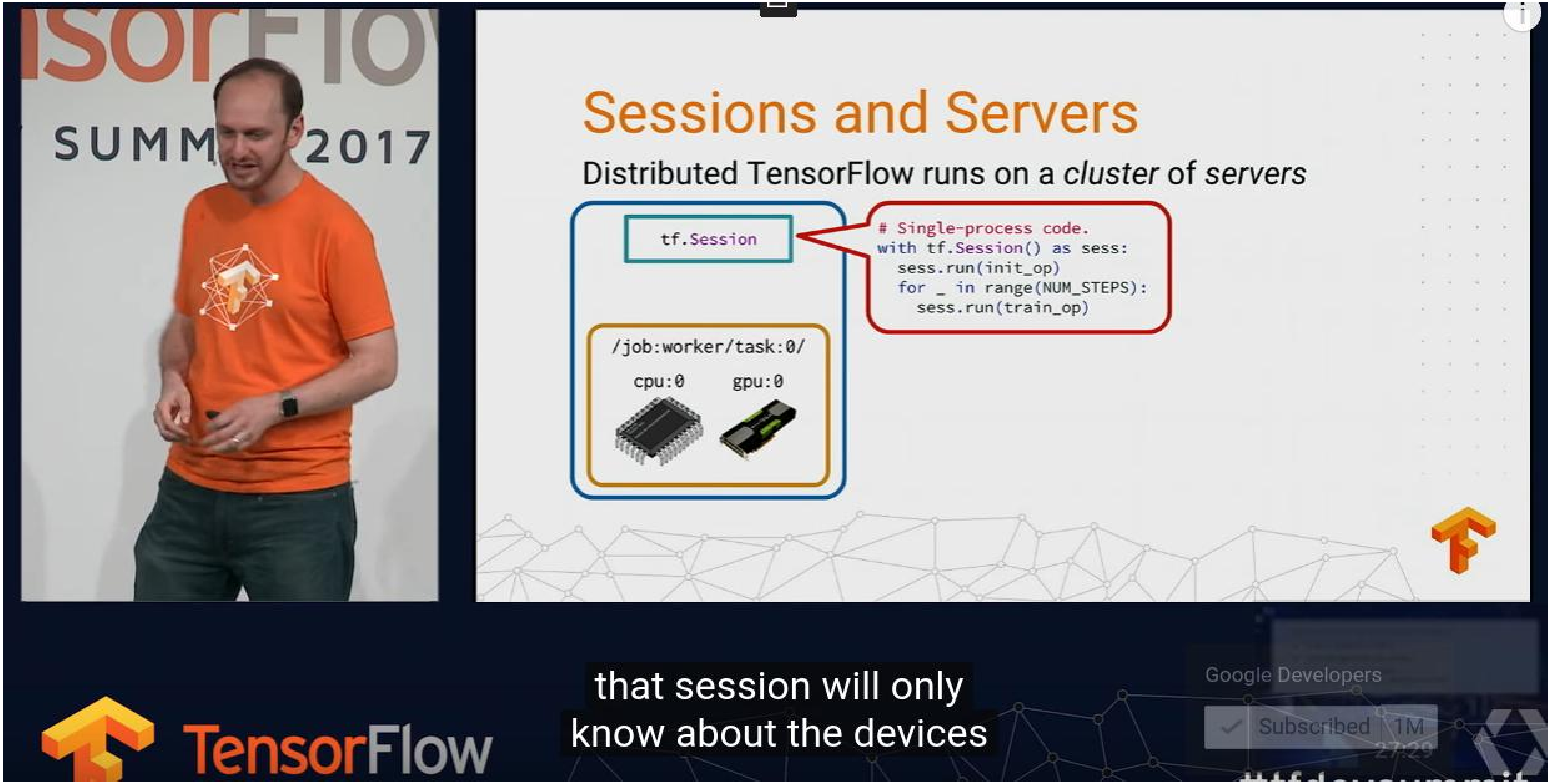

SUMMIT 2017
Experiments and Estimators
High-level APIs package up the whole distributed workflow
def experiment_fn(config, params):
  features = [tf.layers.embedding_column(…),
              tf.layers.bucketized_column(…)]
  return Experiment(
      train_input_fn=…, eval_input_fn=…,
      estimator=DNNClassifier(
          hidden_units=[10, 20], feature_columns=features,
          config, params))
learn_runner.run(experiment_fn, config, …)
uses a recently added class
called Experiment to package up
TensorFlow
DEV SUMMIT 2017
#tfdevsummit

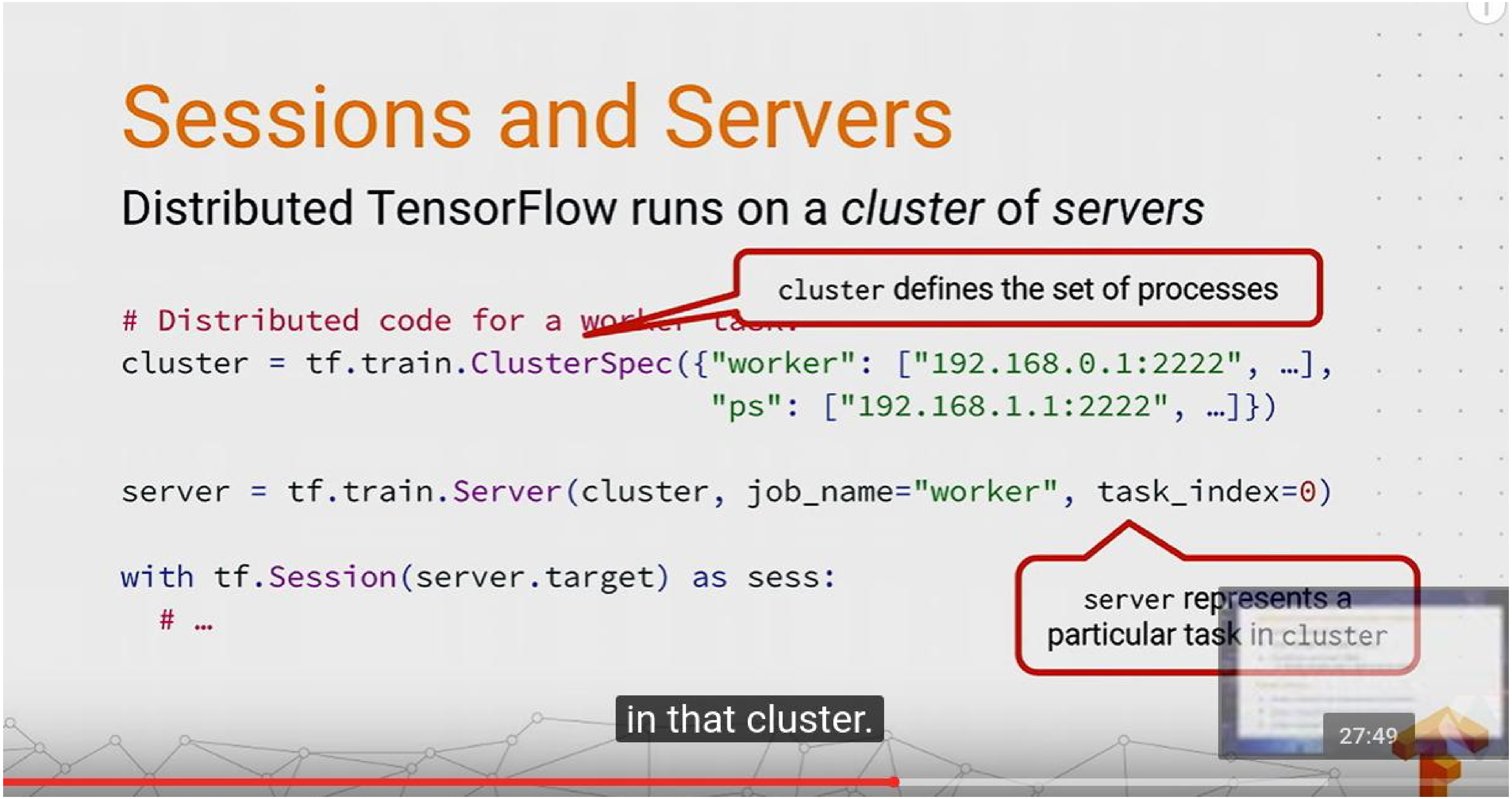
Partitioned variables

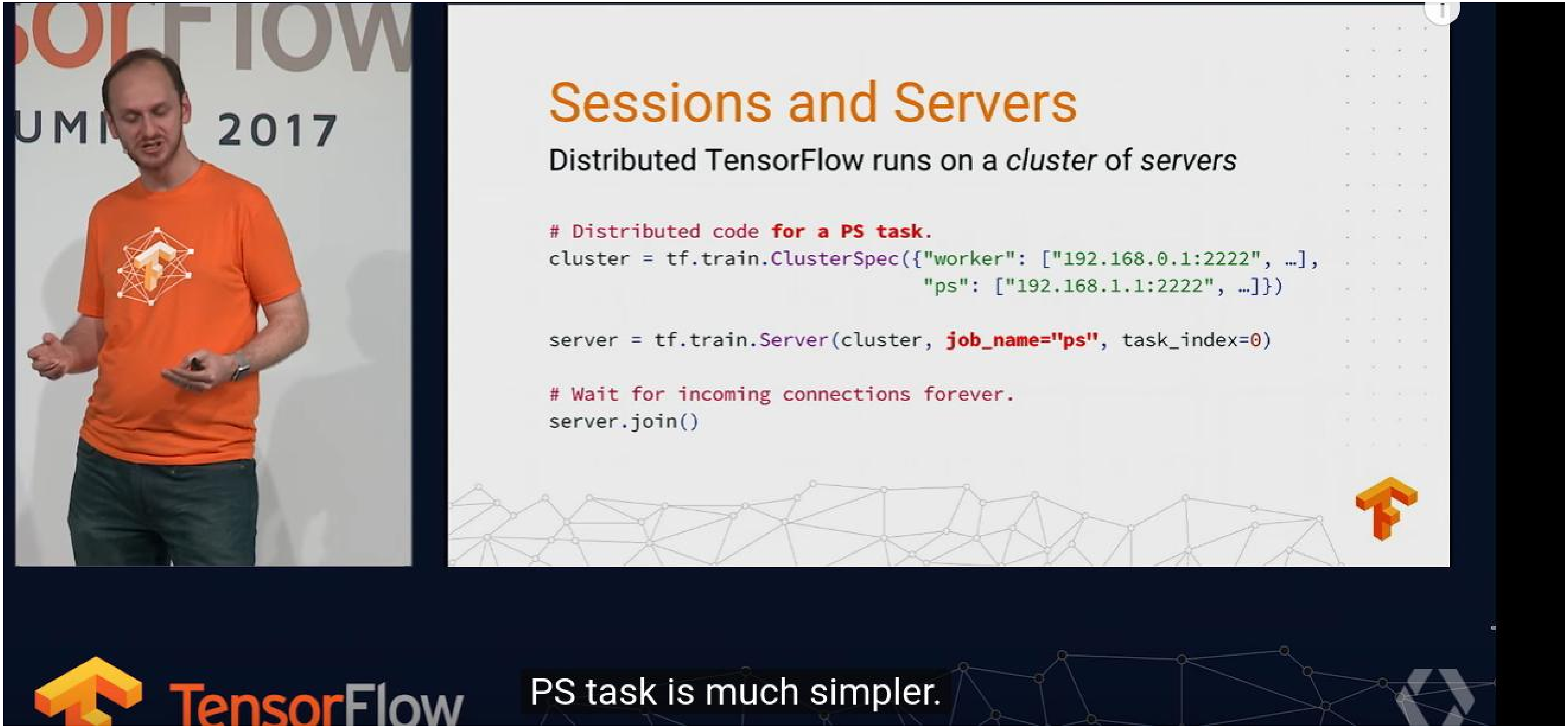
```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(…)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):

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

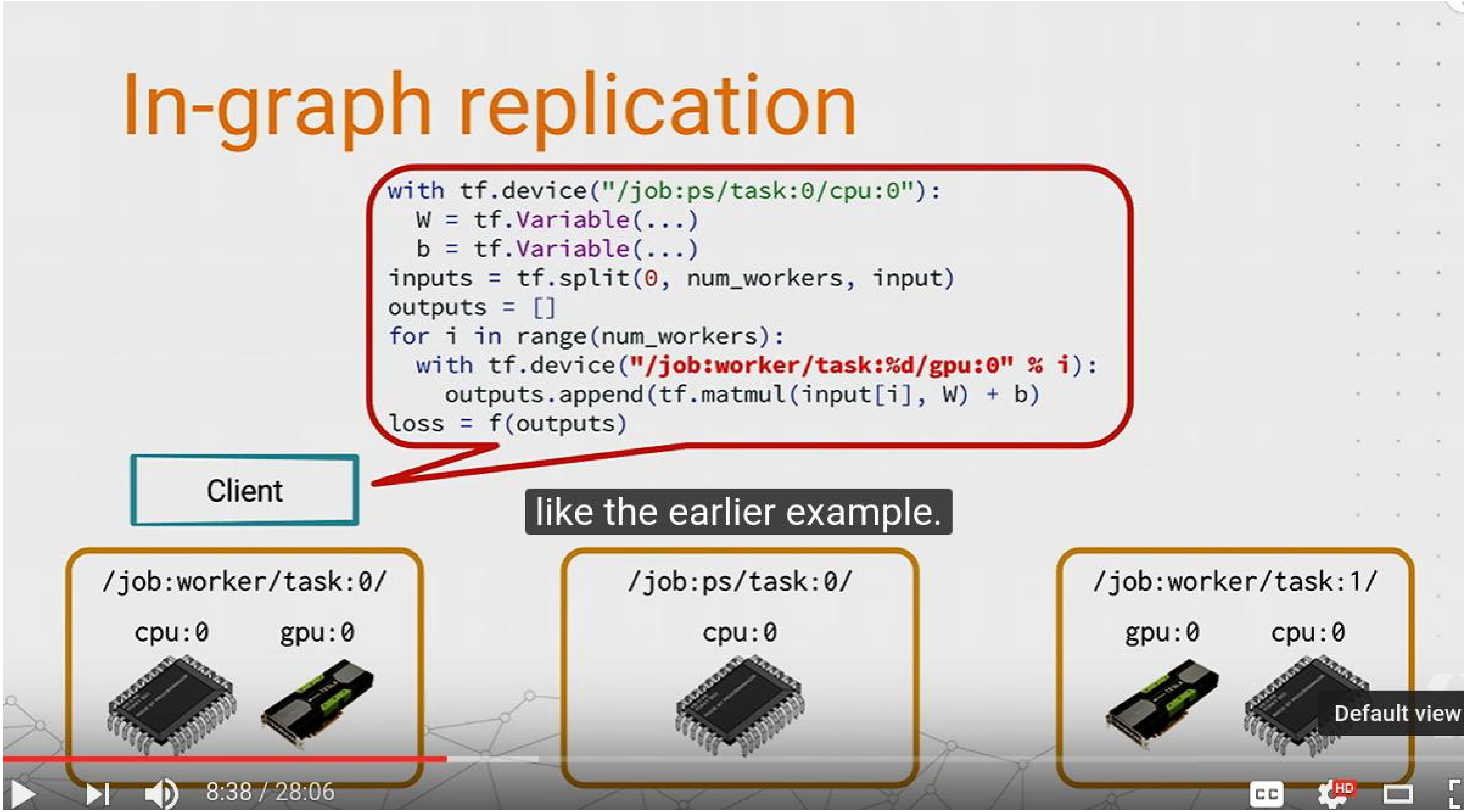
here, TensorFlow will split
the large logical variable




Sessions and Servers
Distributed TensorFlow runs on a cluster of servers
tf.Session
Single-process code.
with tf.Session() as sess:
  sess.run(init_op)
  for _ in range(NUM_STEPS):
    sess.run(train_op)
/job:worker/task:0/
cpu:0
gpu:0
that session will only
know about the devices
TensorFlow
Google Developers


Sessions and Servers
Distributed TensorFlow runs on a *cluster* of *servers*
cluster defines the set of processes
# Distributed code for a wo
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 represents a particular task in cluster
in that cluster.
27:49


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()
2017
TensorFlow
PS task is much simpler.


In-graph replication
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)
Client
like the earlier example.
/job:worker/task:0/
cpu:0
gpu:0
/job:ps/task:0/
cpu:0
/job:worker/task:1/
gpu:0
cpu:0
Default view
8:38 / 28:06

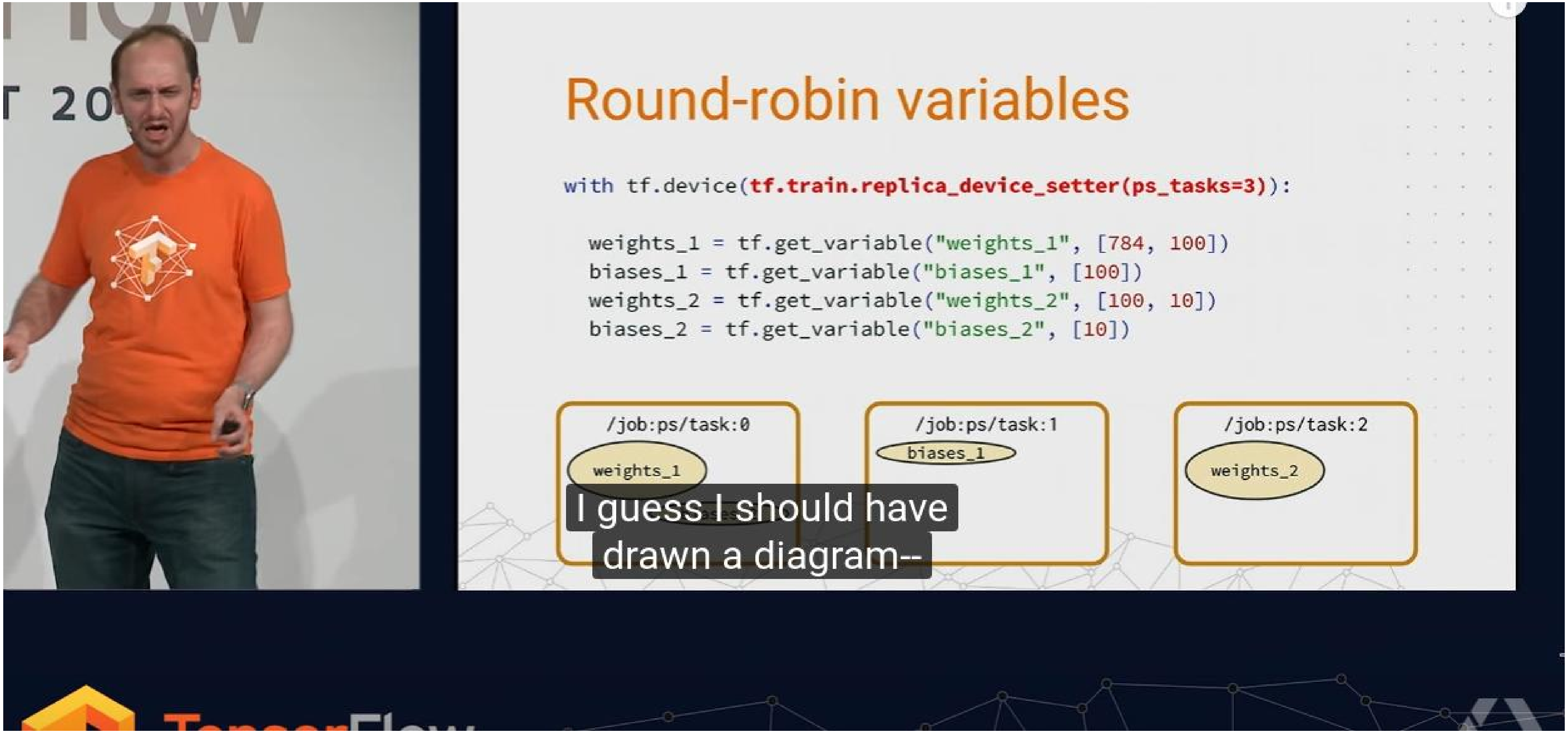
Experiments and Estimators

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))

learn_runner.run(experiment_fn, config, …)
```

Declarative specification of a fully-connected neural network

You tell it how to read
a particular data set


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])
/job:ps/task:0
weights_1
/job:ps/task:1
biases_1
/job:ps/task:2
weights_2
I guess I should have
drawn a diagram--

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])
```

placement strategies.

/job:ps/task:0

/job:ps/task:1

2

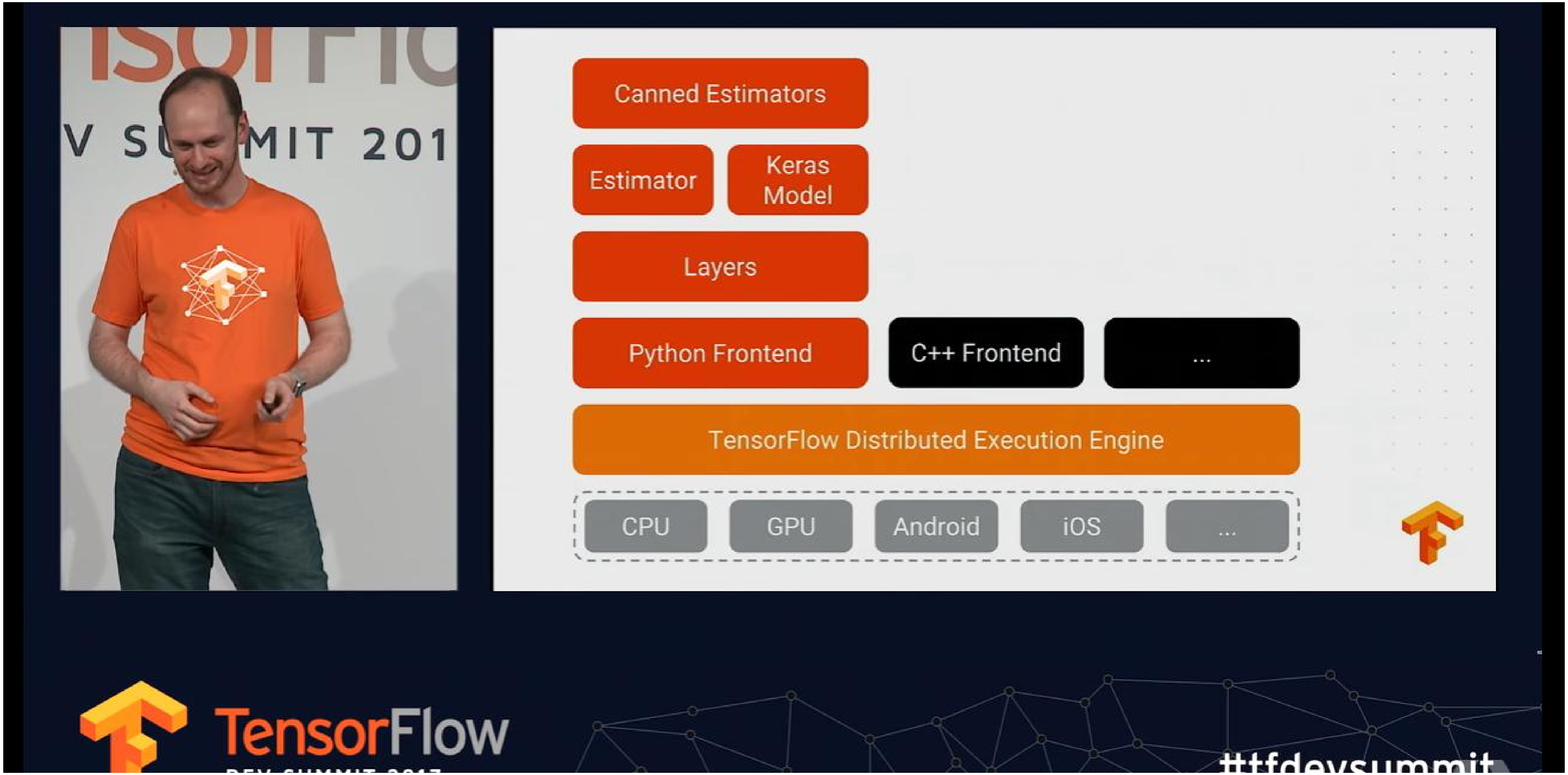
11:53 / 28:06

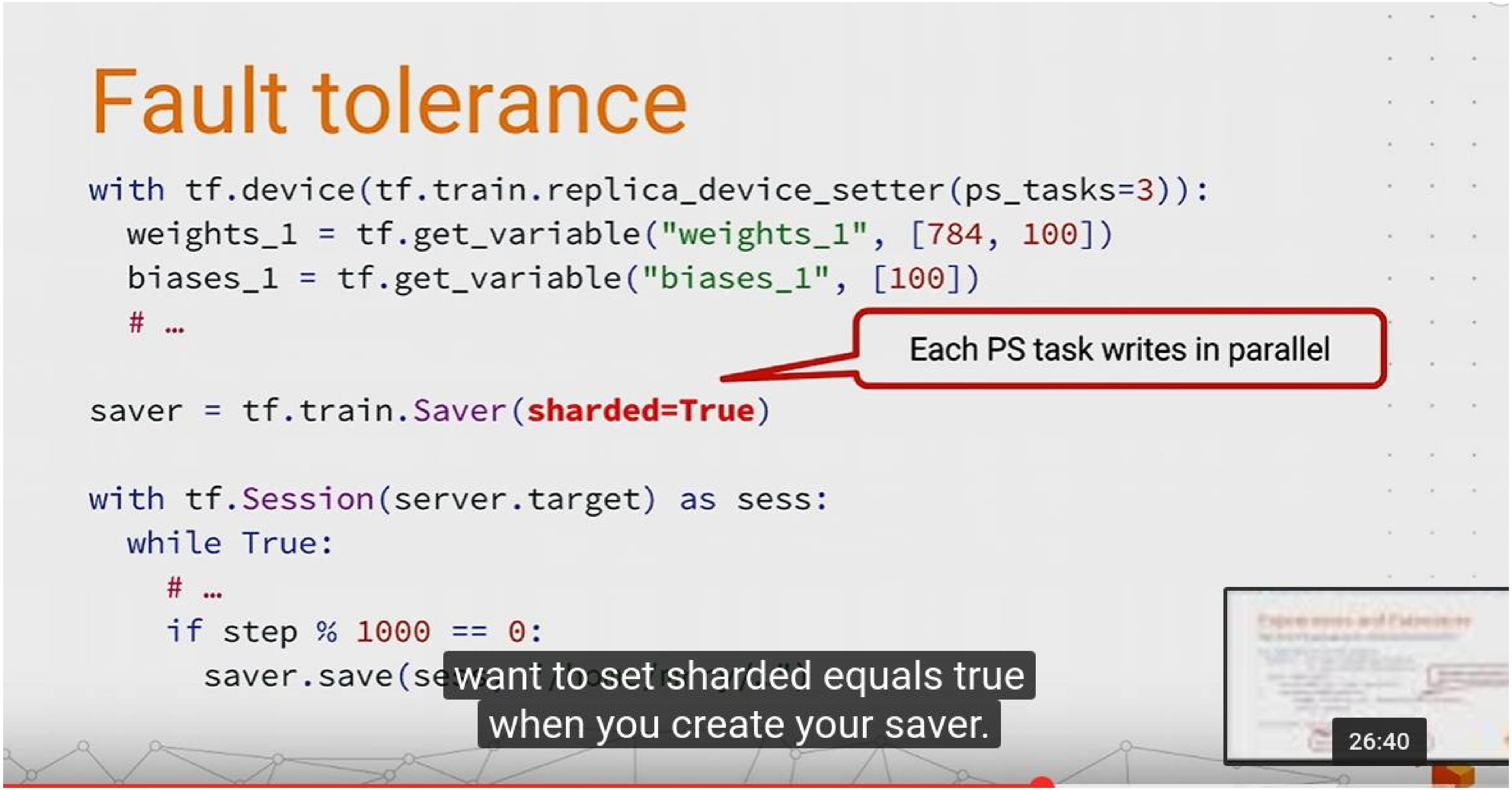
Sessions and Servers

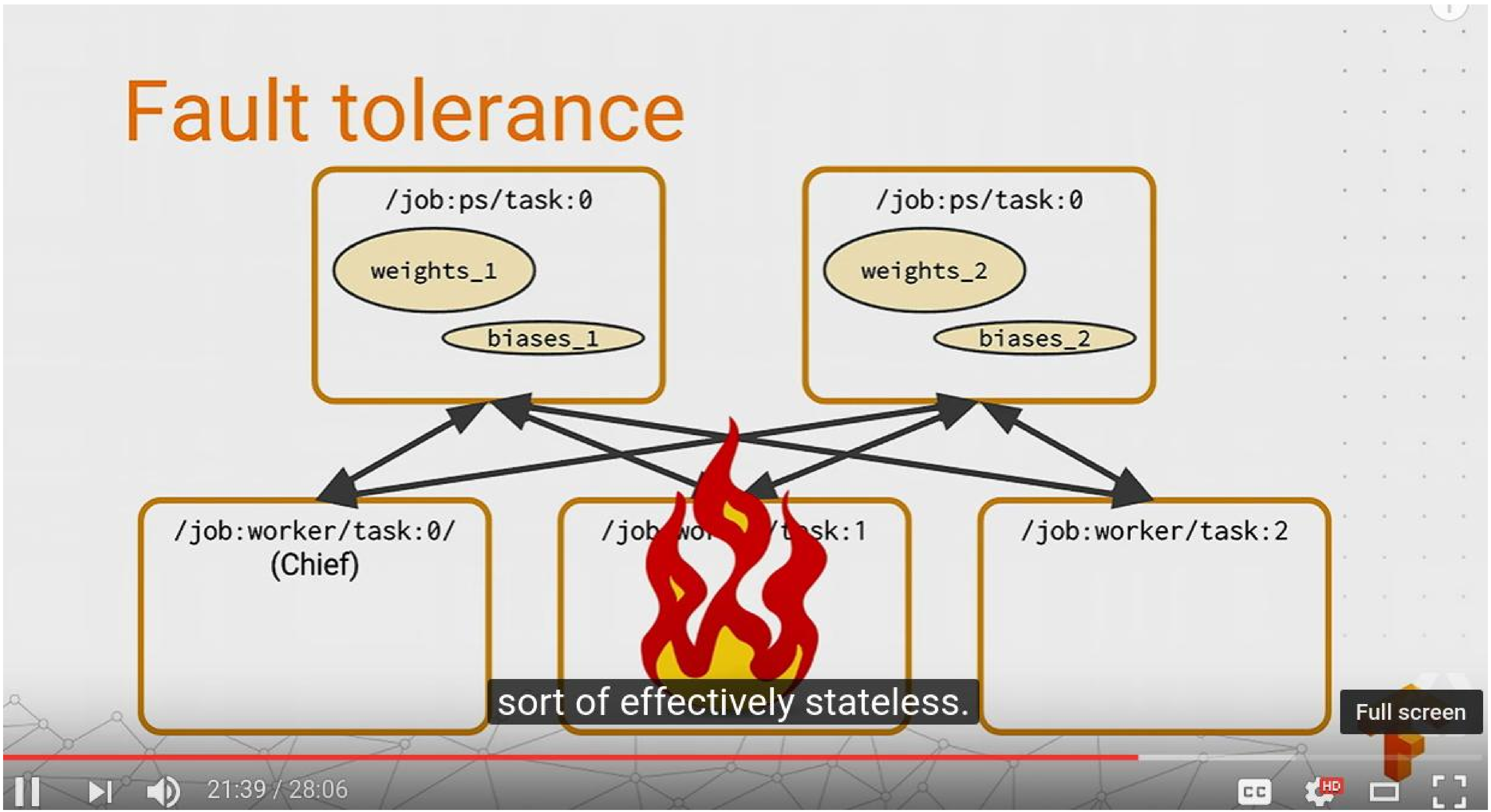
Distributed TensorFlow runs on a *cluster* of *servers*

```
# 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:
  # …
```

the first thing we need to do
is to provide a cluster spec.


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

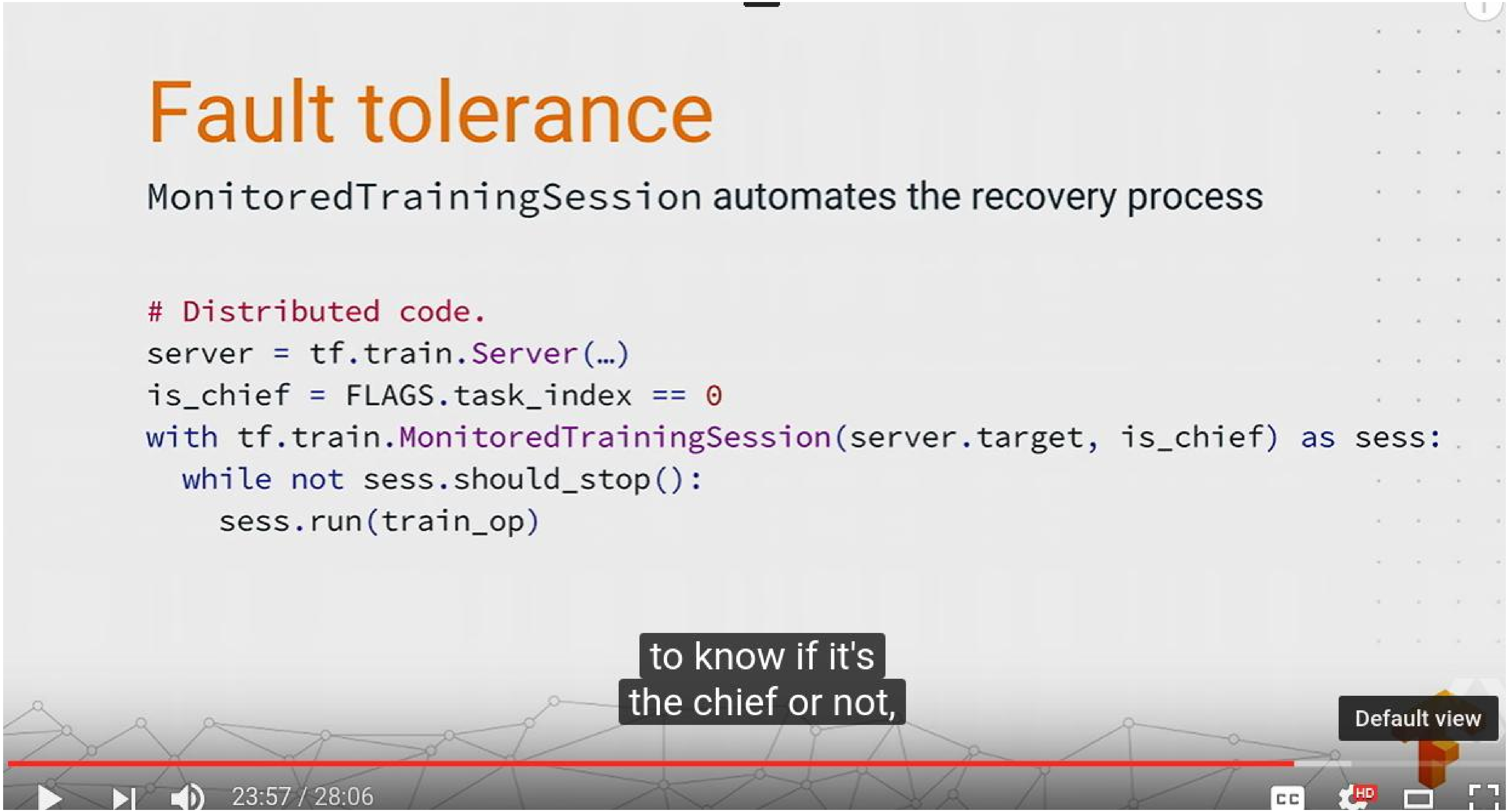

Fault tolerance
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):
weights_1 = tf.get_variable("weights_1", [784, 100])
biases_1 = tf.get_variable("biases_1", [100])
# …
saver = tf.train.Saver(sharded=True)
Each PS task writes in parallel
with tf.Session(server.target) as sess:
while True:
# …
if step % 1000 == 0:
saver.save(se
want to set sharded equals true
when you create your saver.
26:40


Fault tolerance
/job:ps/task:0
weights_1
biases_1
/job:ps/task:0
weights_2
biases_2
/job:worker/task:0/
(Chief)
/job:worker/task:2
sort of effectively stateless.
Full screen
21:39 / 28:06

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/…")
```

parameters to disk.


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)
to know if it's
the chief or not,
Default view
23:57 / 28:06
HD

Sessions and Servers

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

`cluster` defines the set of processes

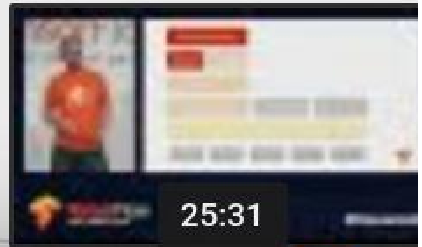
```
# Distributed code for a worker task:
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

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

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

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)
```

before you start training.



Device placement

`tf.train.replica_device_setter()`

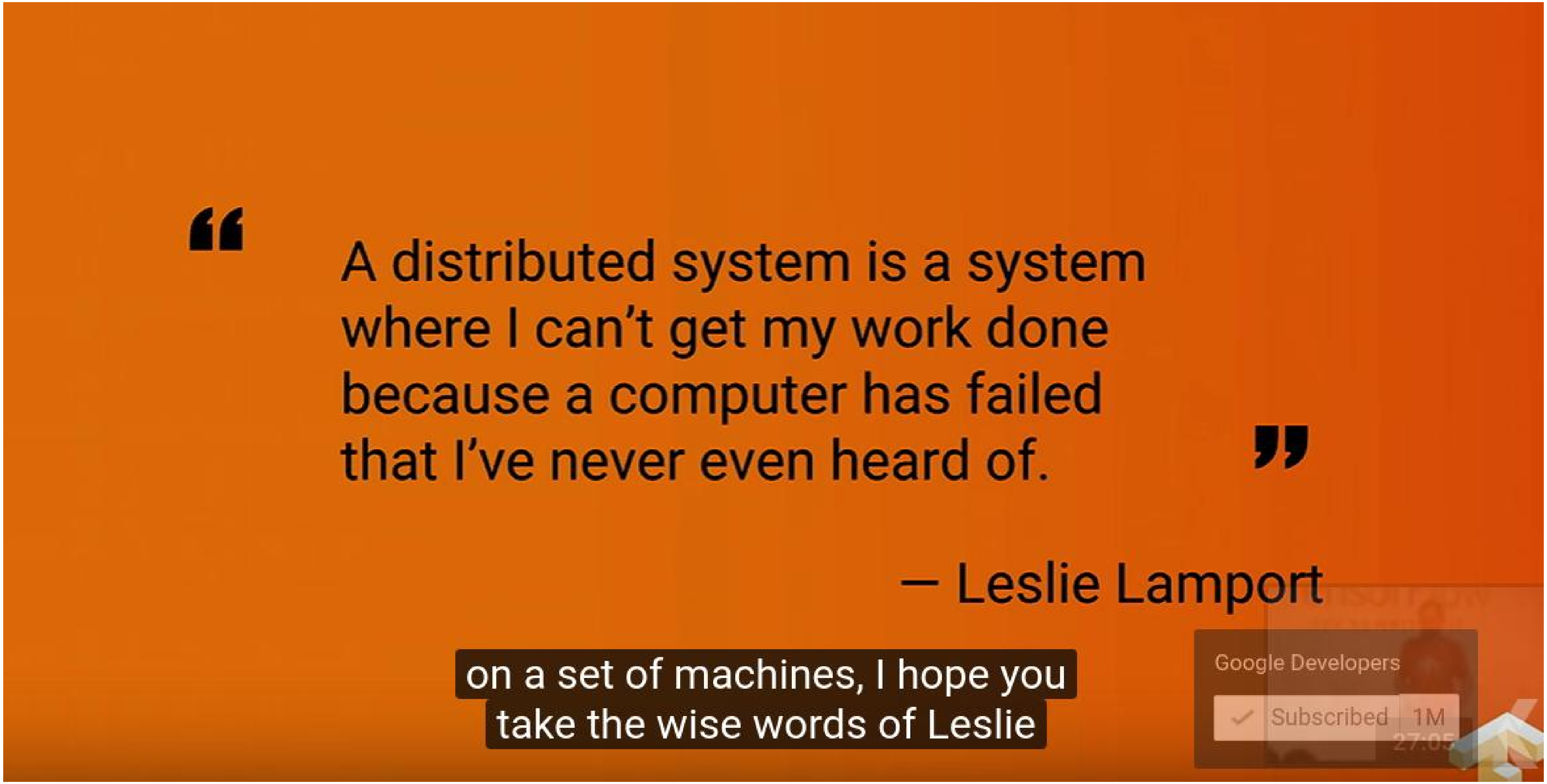
Simple heuristic for between-graph partitioning

- 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

Variable placement

```
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])
```

So far, I've just been
putting them on jobPS task 0


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
on a set of machines, I hope you
take the wise words of Leslie
Google Developers
Subscribed
1M