慕课学院
www.mooc.ai

计算图和 TensorFlow 基础

讲师：白发川

简介：高级数据架构师

邮箱：fcbai@thoughtworks.com

慕课学院
www.mooc.ai

学习目标：

- 理解TF的基本概念和结构
- 理解TF的基本操作
- 能够使用TF编写简单的程序

讲师介绍：

ThoughtWorks 数据架构师，深度学习框架 deeplearning.scala 贡献者。

设计实现了金融、工业、互联网等多个领域的大数据平台建设和数据处理。

ThoughtWorks®

大数据&人工智能

TENSORFLOW概述

什么是TF

- 一个使用数据流图的开源数值计算软件库
- Google研发的一个深度神经网络框架
- 可以应用在多个领域

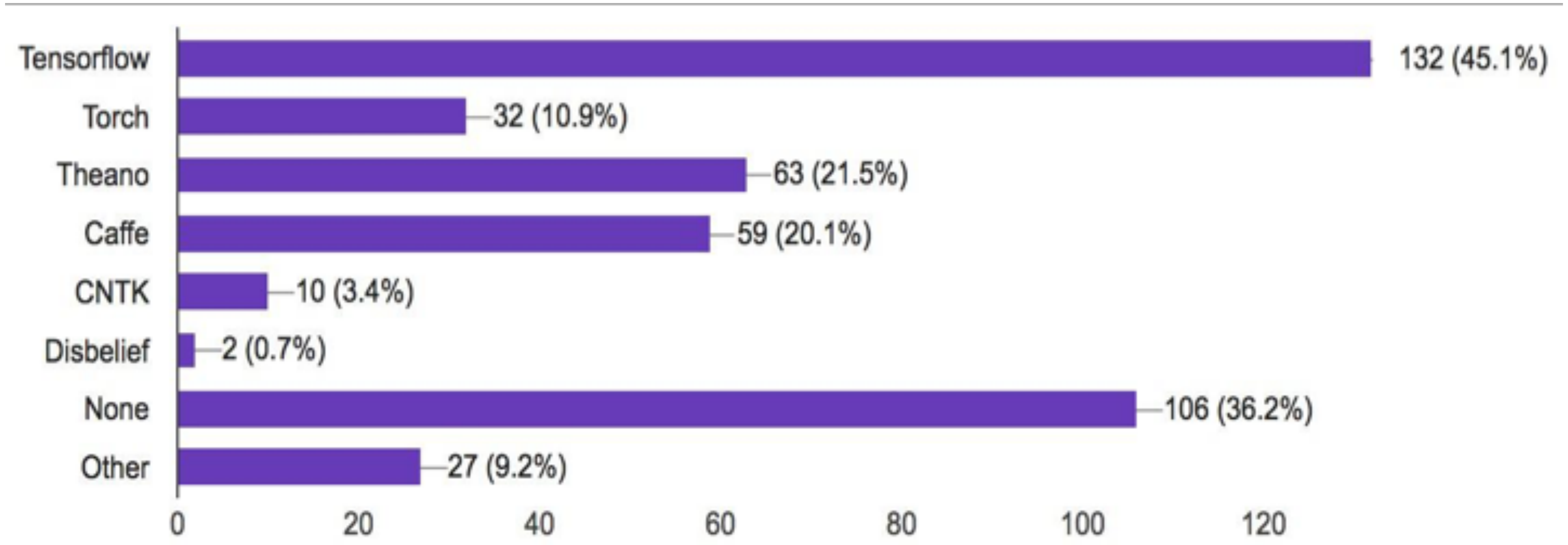
开始使用TF

```
pip install tensorflow

import tensorflow as tf
```

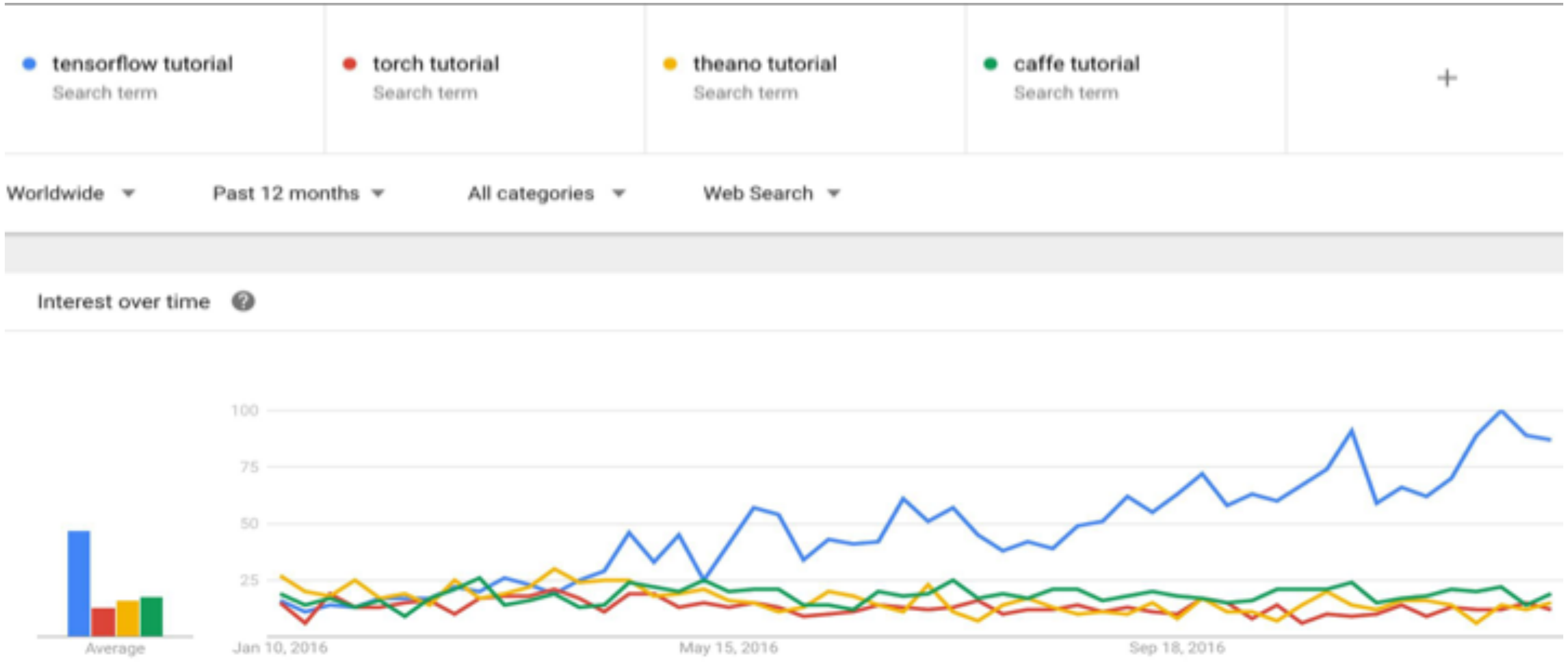
工具包

- TF Learn
- TF Slim
- 更高级的API封装：Keras，Pretty Tensor

TF不是唯一的


Tensorflow
Torch
Theano
Caffe
CNTK
Disbelief
None
Other
132 (45.1%)
32 (10.9%)
63 (21.5%)
59 (20.1%)
10 (3.4%)
2 (0.7%)
106 (36.2%)
27 (9.2%)
0
20
40
60
80
100
120

TF增长率

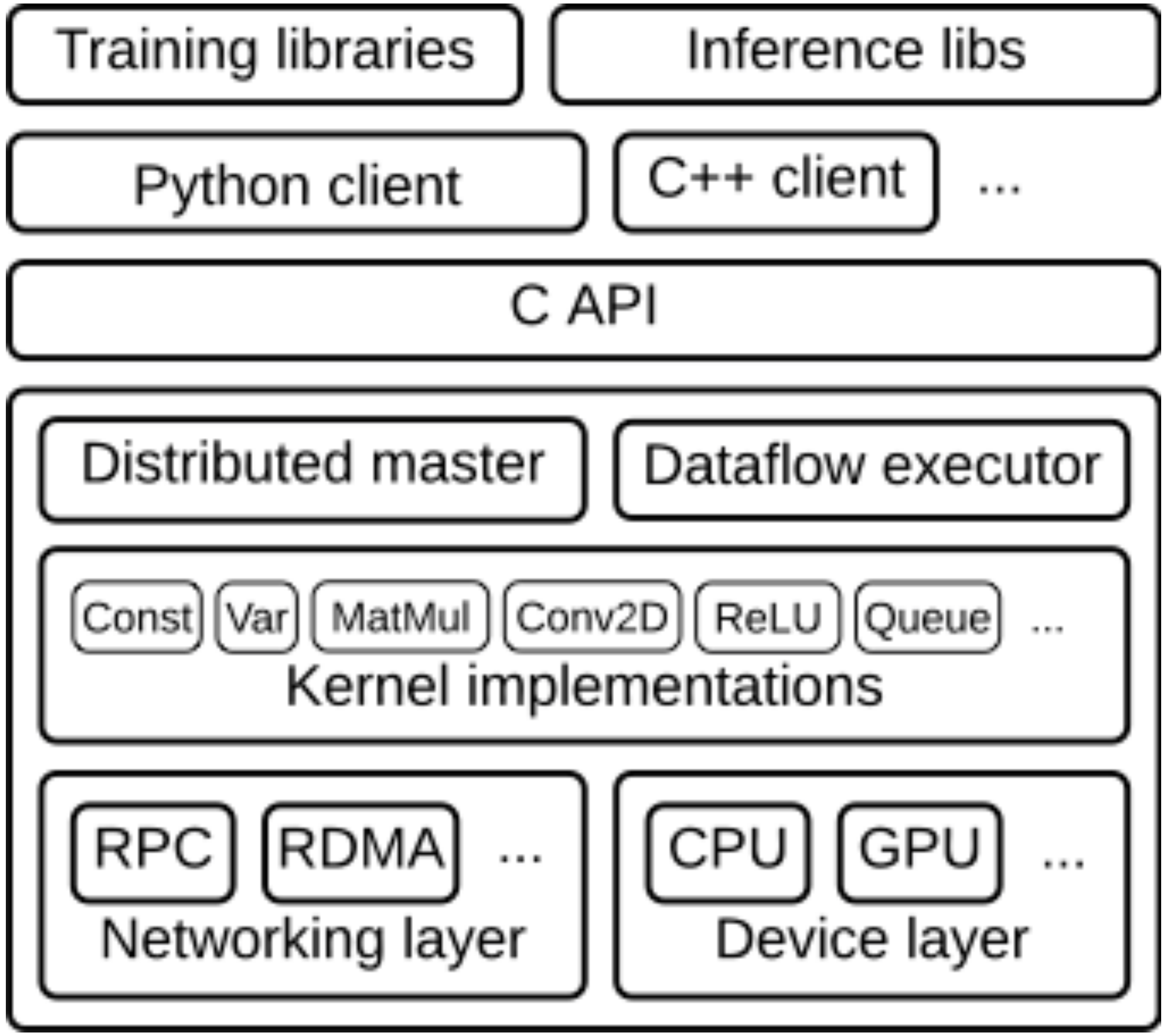


- tensorflow tutorial
  Search term
- torch tutorial
  Search term
- theano tutorial
  Search term
- caffe tutorial
  Search term

Worldwide ▾ Past 12 months ▾ All categories ▾ Web Search ▾

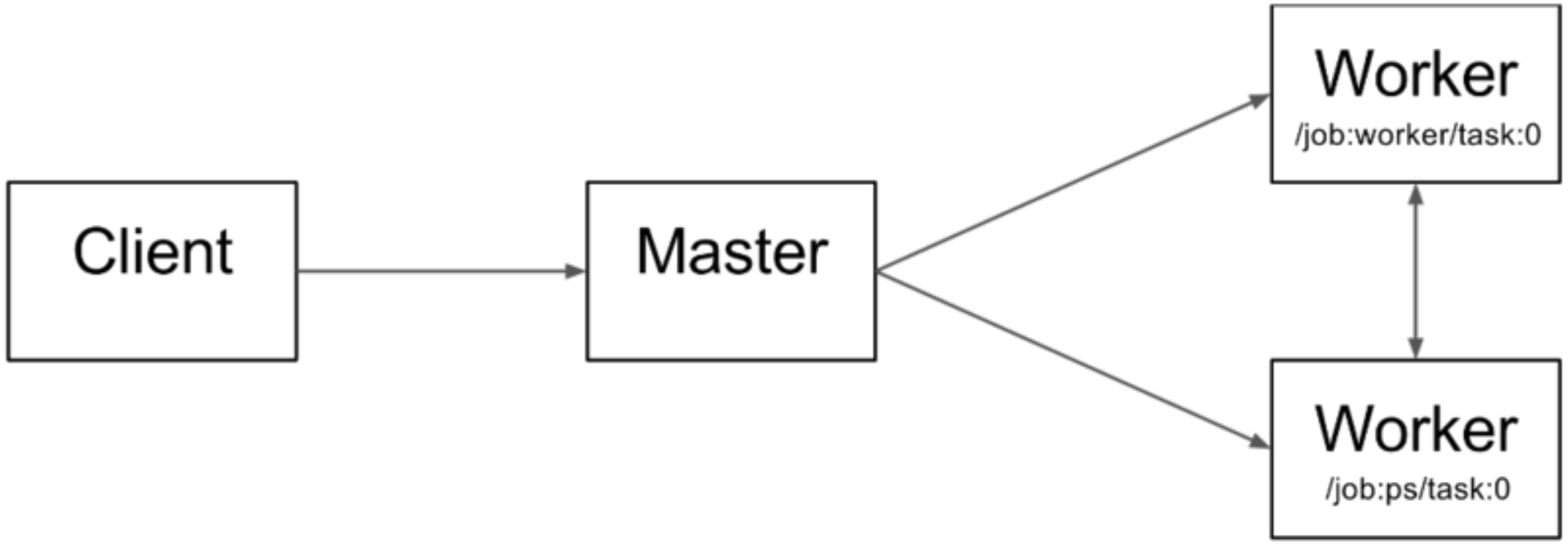
Interest over time

TF架构



- RPC和RDMA为网络层，主要负责传递神经网络算法参数。
- CPU和GPU为设备层，主要负责神经网络算法中具体的运算操作。
- Kernel为TensorFlow中算法操作的具体实现，如卷积操作，激活操作等。
- Distributed Master用于构建子图；切割子图为多个分片，不同的子图分片运行在不同的设备上；Master还负责分发子图分片到Executor/Work端。Executor/Work在设备（CPUs，GPUs，etc.）上，调度执行子图操作；并负责向其它Worker发送和接收图操作的运行结果。
- C API把TensorFlow分割为前端和后端，前端（Python/C++/Java Client）基于C API触发TensorFlow后端程序运行。
- Training libraries和Inference libs是模型训练和推导的库函数，为用户开发应用模型使用。

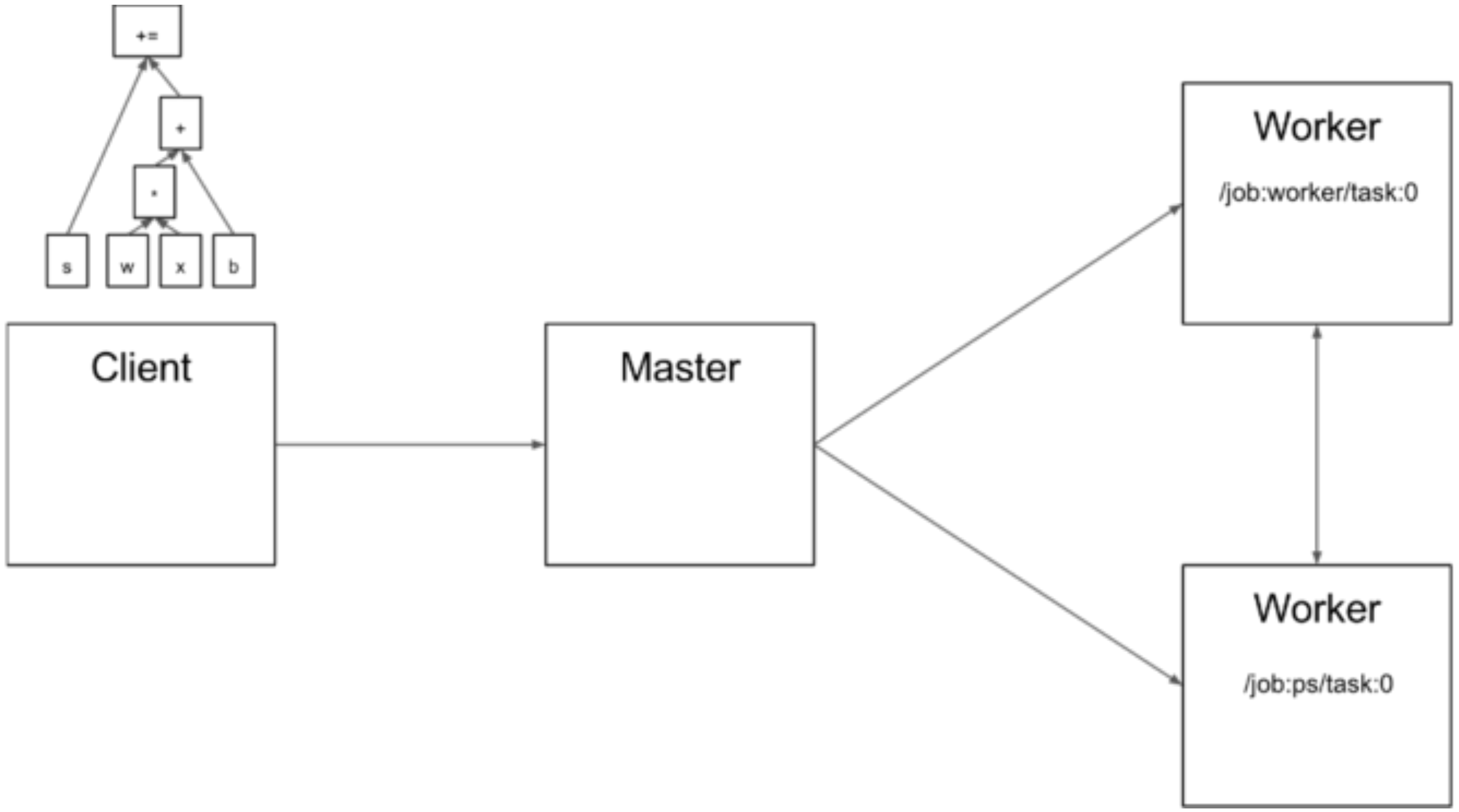
内部工作原理



- "/job:worker/task:0" 和 "/job:ps/task:0" 表示worker中的执行服务。
- “job:ps"表示参数服务器，用于存储及更新模型参数。
- “job:worker"用于优化模型参数，并发参数发送到参数服务器上。
- Distributed Master和Worker Service只存在于分布式TensorFlow中。
- 单机版本的TensorFlow实现了Local的Session，通过本地进程的内部通讯实现上述功能。

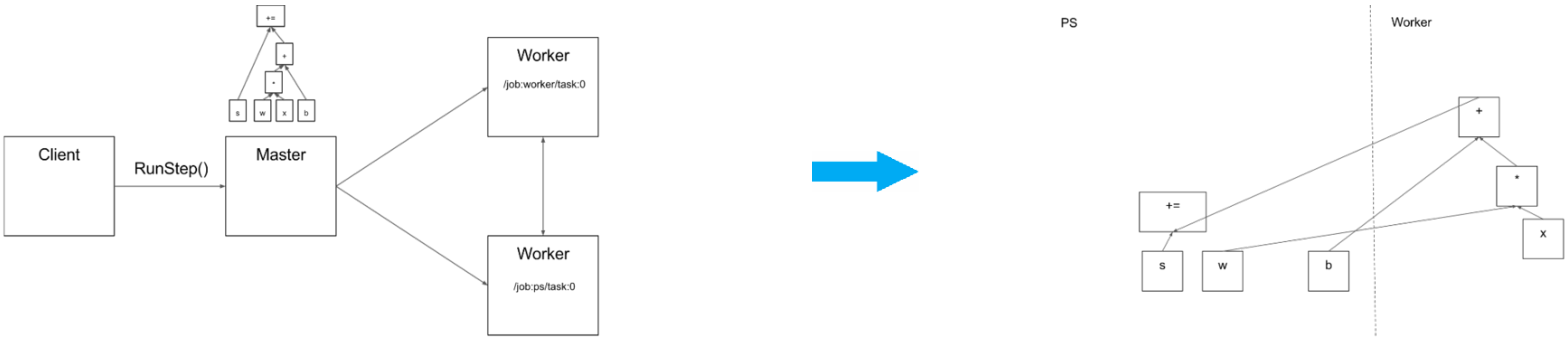
程序编写

s+=w*x+b



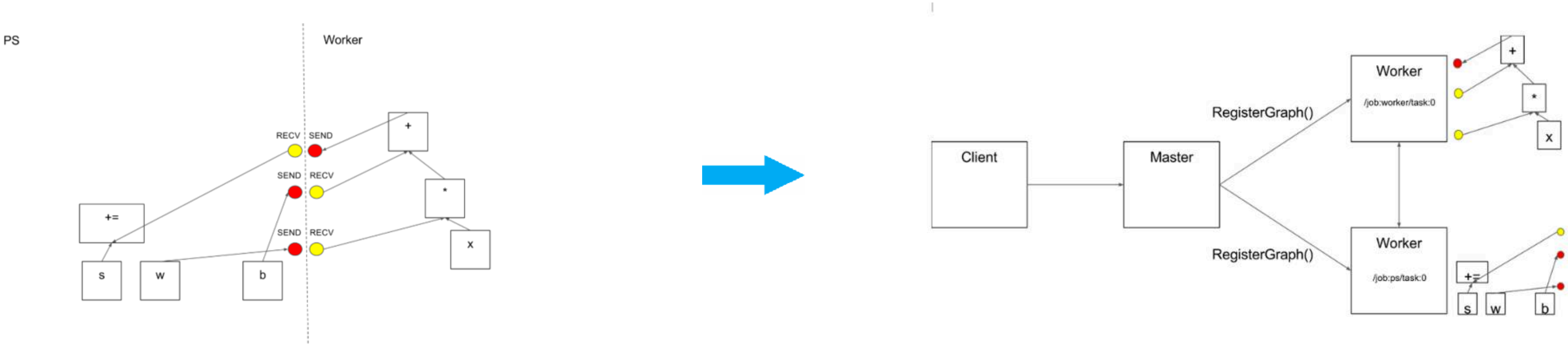
用户编写TensorFlow应用程序生成计算图，Client组件会创建Session，并通过序列化技术，发送图定义到Distributed Master组件。

子图切割



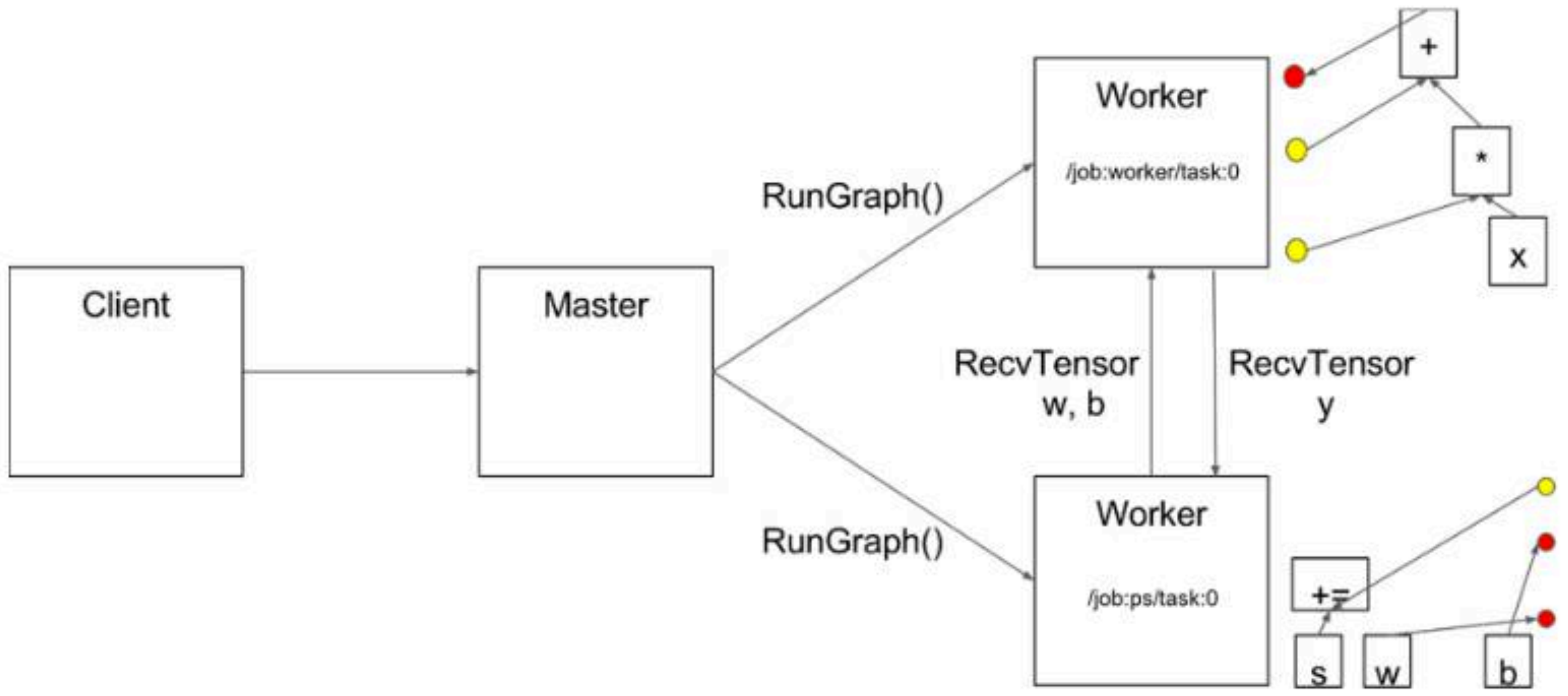
当Client触发Session运算的时候，Maser构建将要运行的子图。并根据设备情况，切割子图为多个分片。

并行训练



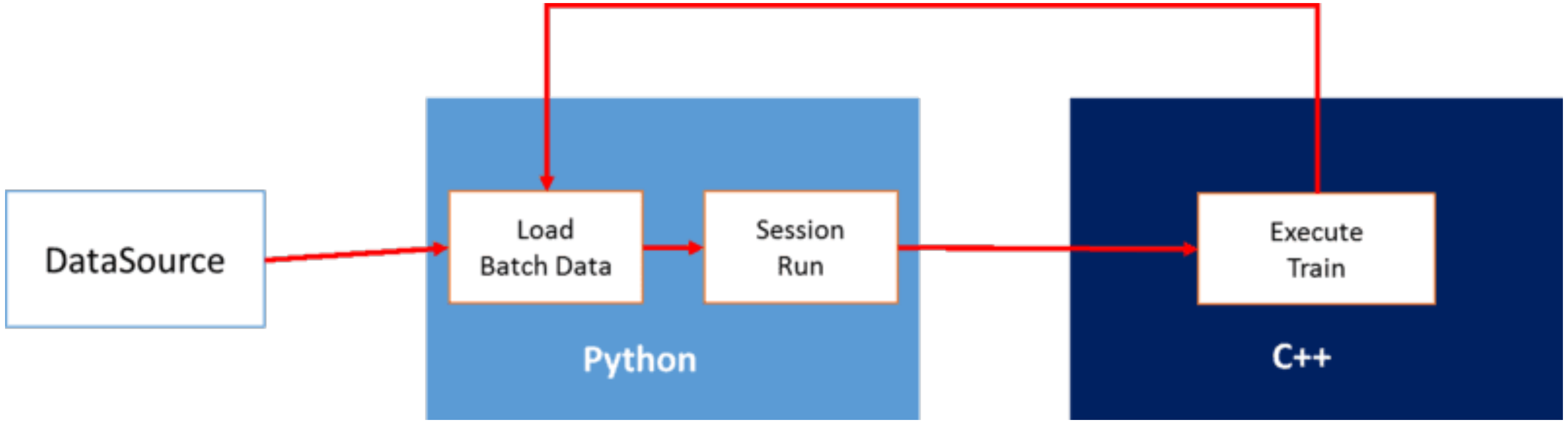
Distributed Master会根据模型参数的分区情况进行切割边，在Task间插入发送和接收Tensor信息的通信节点
接着Distributed Master通过RegisterGraph方法发送子图分片给Task

并行训练

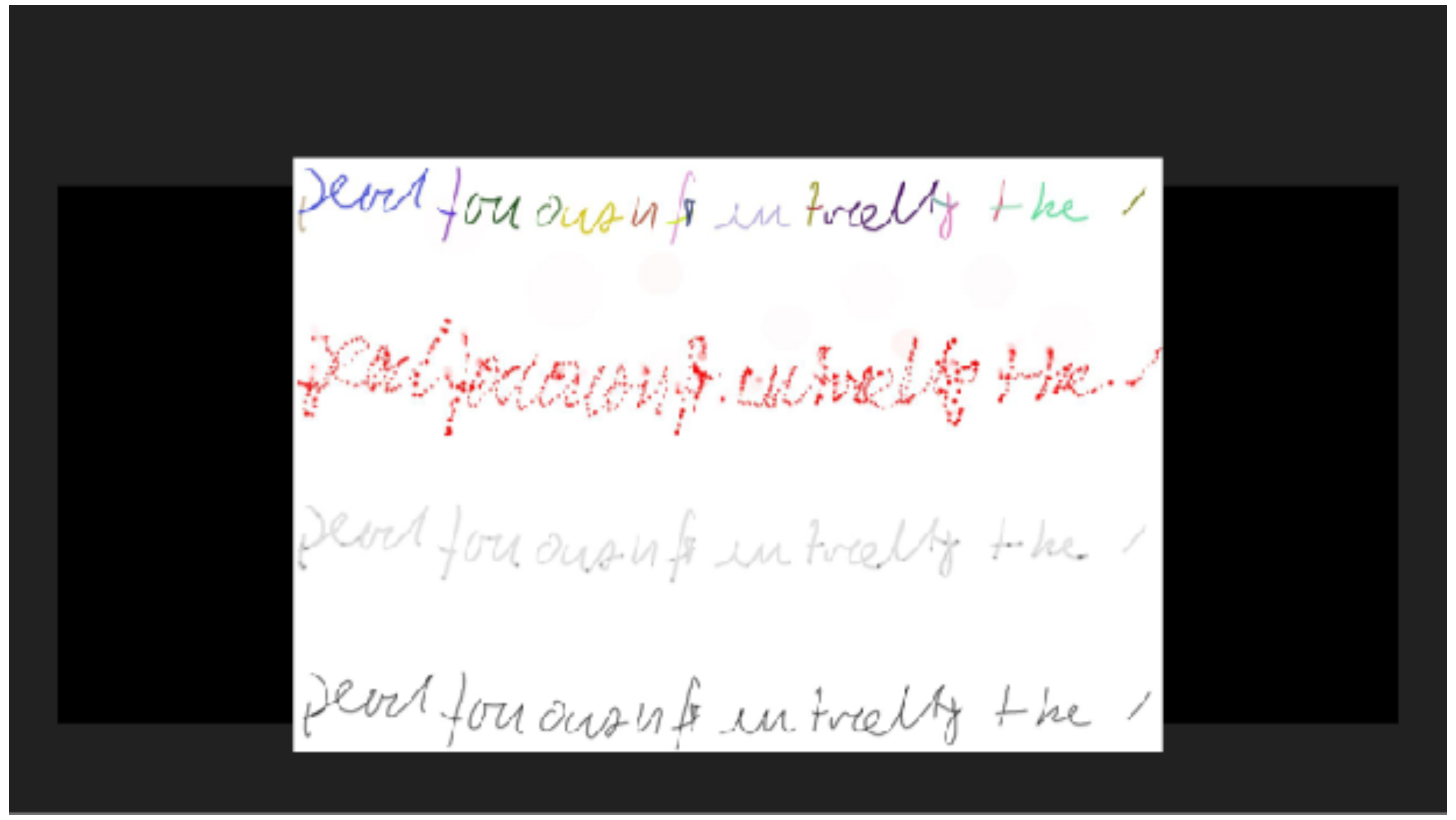


Master通过RunGraph触发子图运算，Worker会使用GPU/CPU运算设备执行TensorFlow Kernel运算。在本节点的CPU和GPU之间，使用cudaMemcpyAsync传输数据；在本节点GPU和GPU之间，使用peer-to-peer DMA传输数据，避免通过CPU复制数据。TensorFlow使用gRPC（TCP）和RDMA （Converged Ethernet）技术，实现Worker间的数据通信及传输

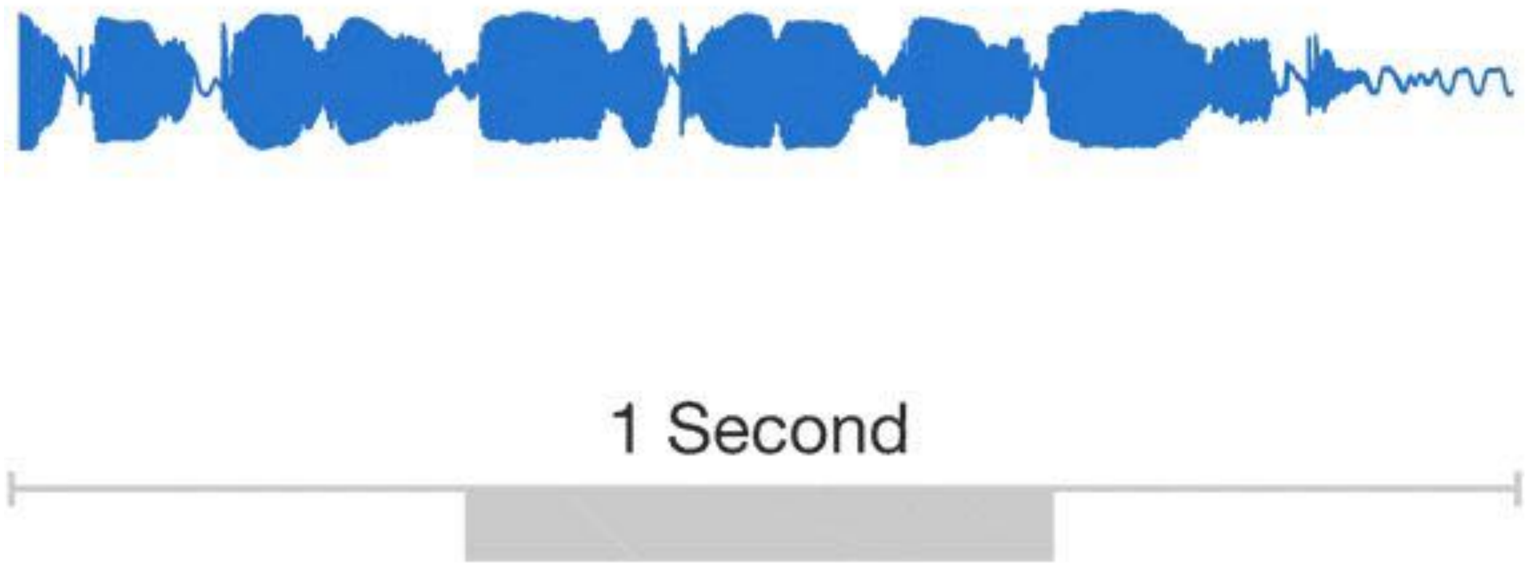
程序设计


DataSource
Load
Batch Data
Session
Run
Python
Execute
Train
C++

TF做的好玩的东西



语音转文字



ThoughtWorks®

大数据&人工智能

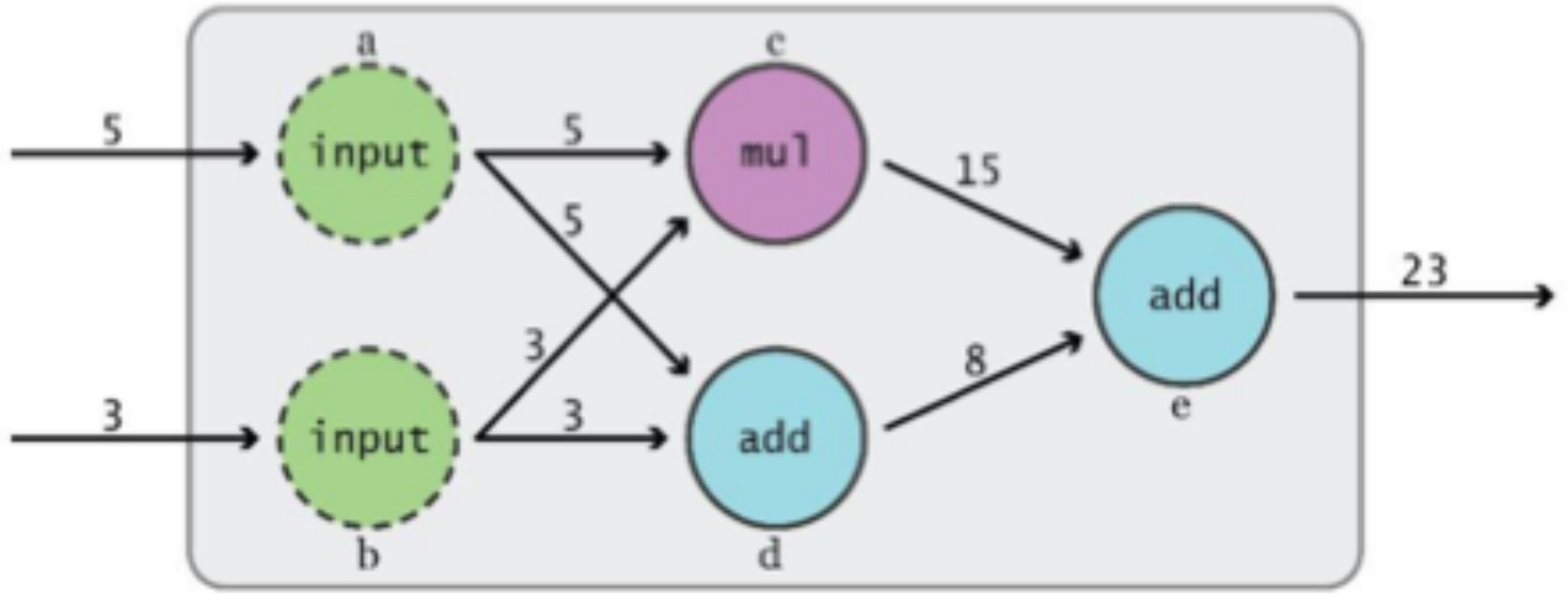
计算图和SESSION

什么是数据流图

数据流图用“结点”（nodes）和“线”(edges)的有向图来描述数学计算。“节点”
一般用来表示施加的数学操作，但也可以表示数据输入（feed in）的起点/输出（push
out）的终点，或者是读取/写入持久变量（persistent
variable）的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size可动态调整”的多维数据数组，即“张量”（tensor）。张量从图中流过的直观图像是这个工具取名为“Tensorflow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。

数据流图

- 构建数据流图
- 使用一个Session来执行图中的操作

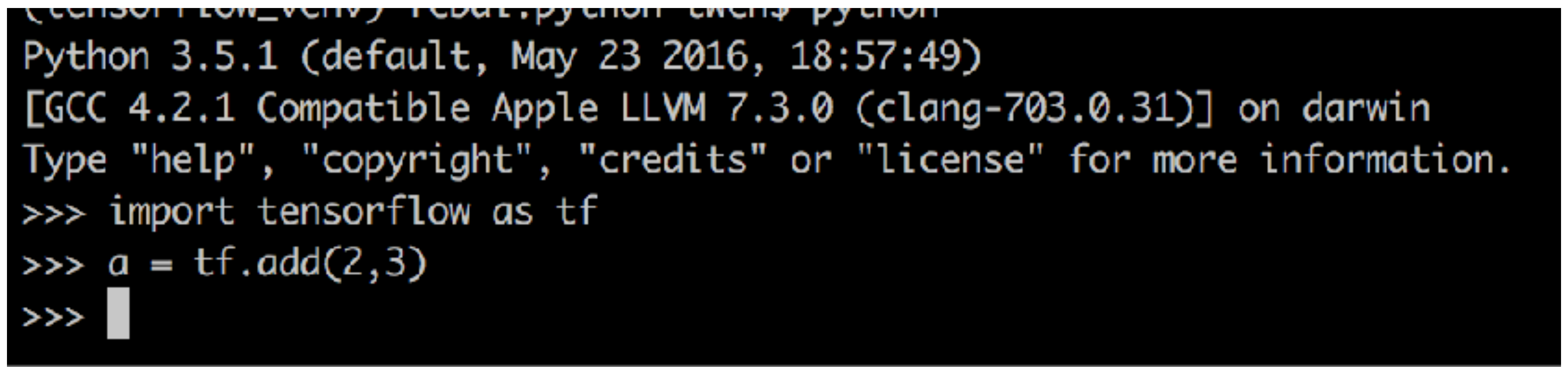


TENSOR

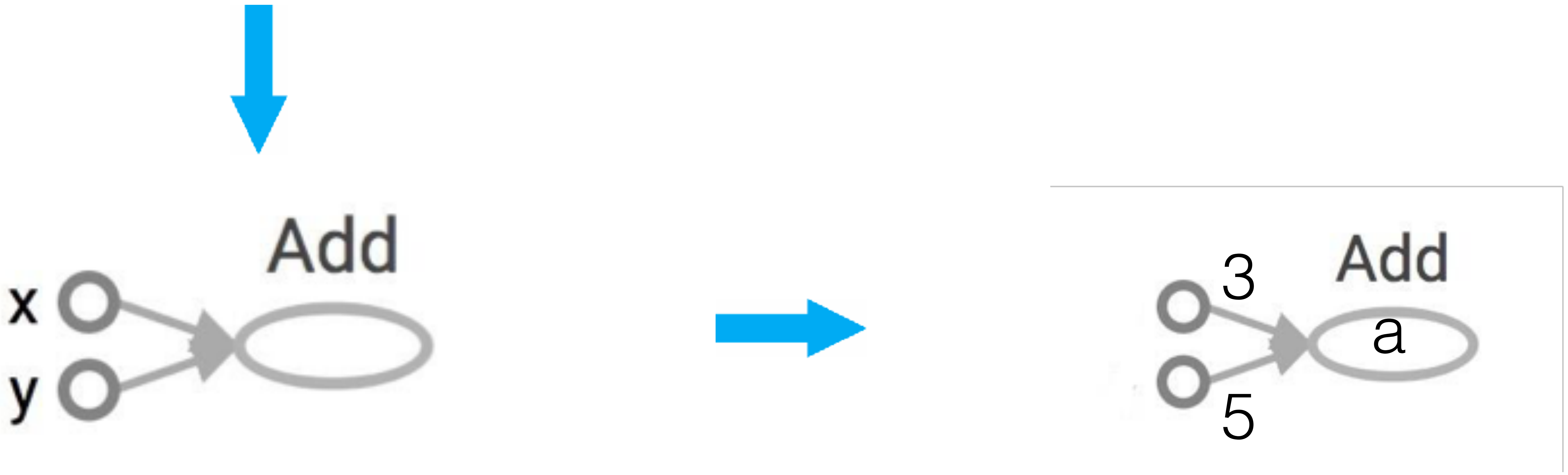
Tensor：一个N维的矩阵

- 0维：数值
- 1维：向量
- 2维：矩阵
- 以及更多

数据流图

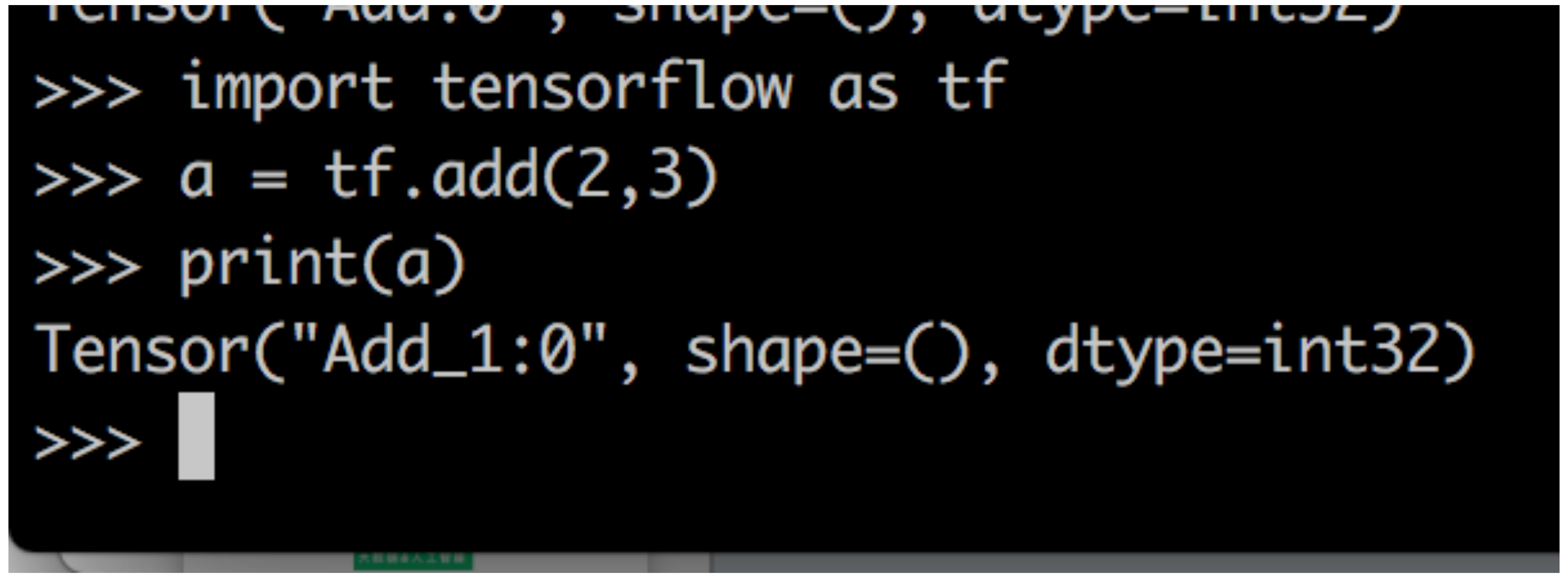


```
(tensorflow_venv) rebat:python twch$ python
Python 3.5.1 (default, May 23 2016, 18:57:49)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> a = tf.add(2,3)
>>> 
```

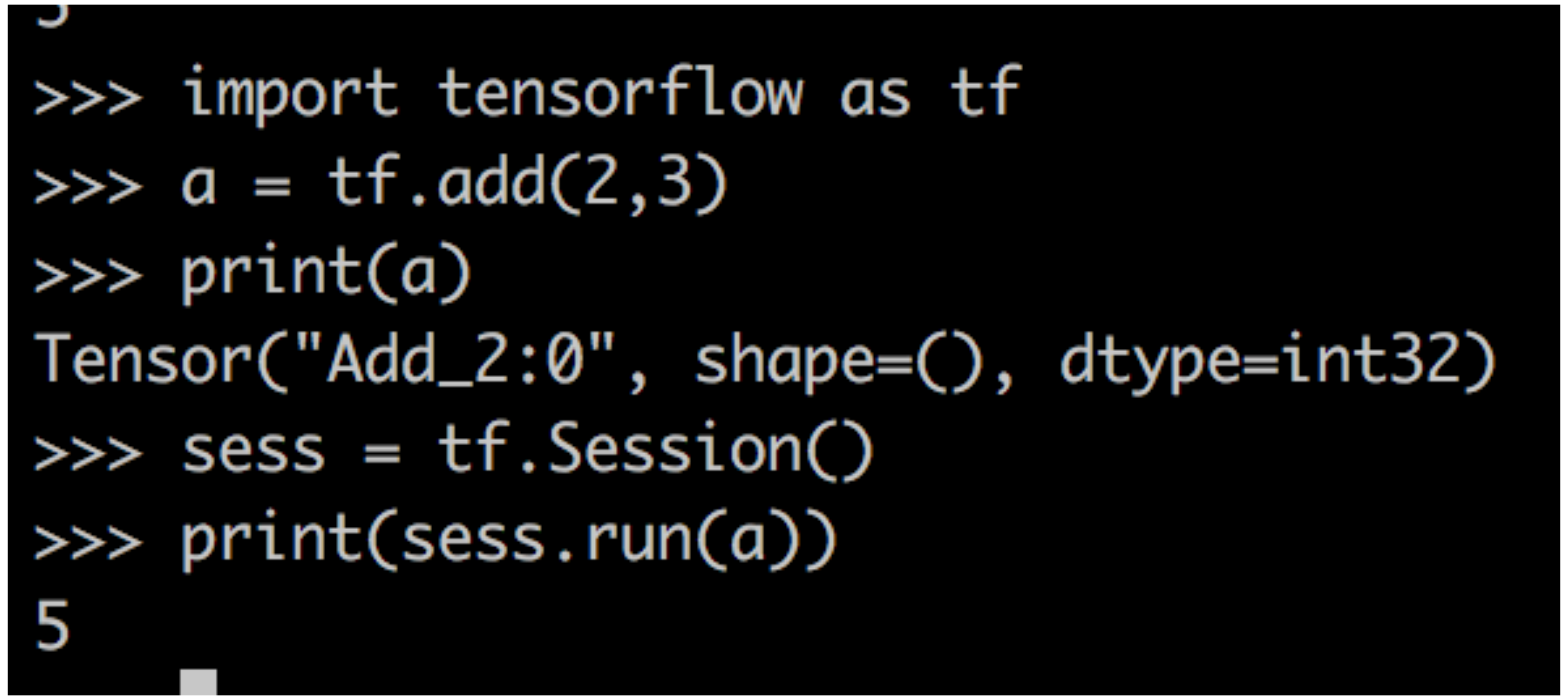


为什么是x，y?

如何获取变量的值

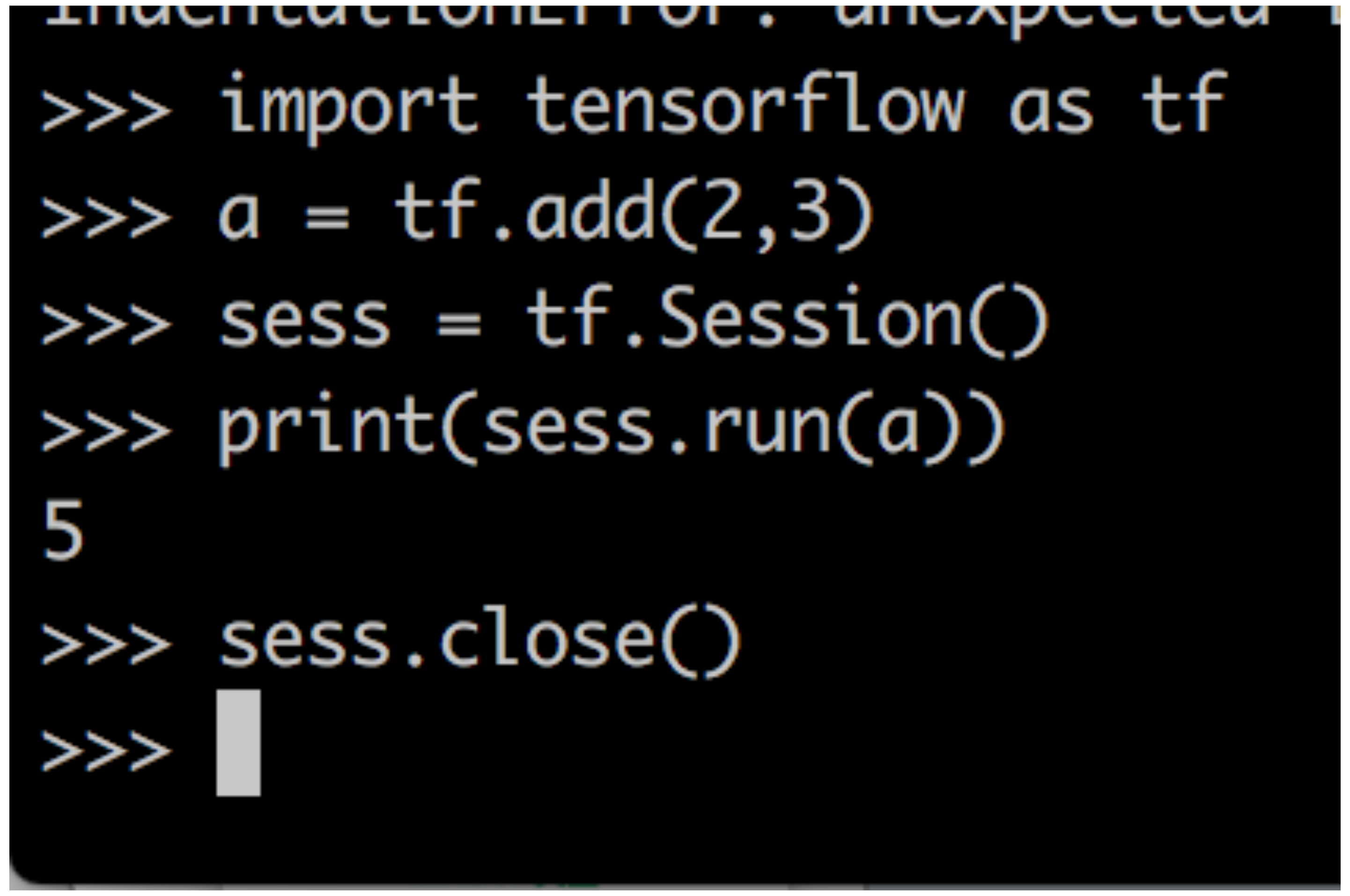


```
Tensor("Add:0", shape=(), dtype=int32)
>>> import tensorflow as tf
>>> a = tf.add(2,3)
>>> print(a)
Tensor("Add_1:0", shape=(), dtype=int32)
>>> 
```



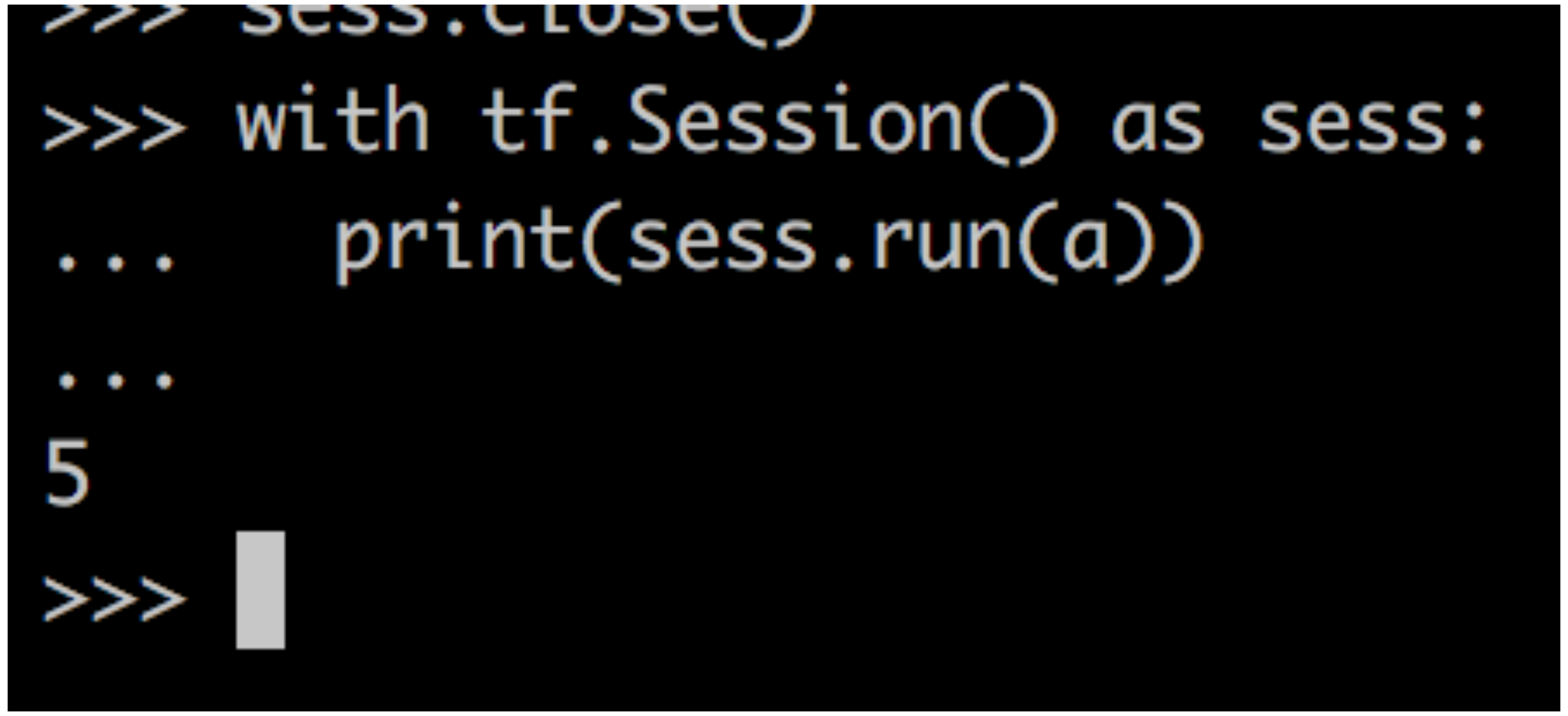
```
5
>>> import tensorflow as tf
>>> a = tf.add(2,3)
>>> print(a)
Tensor("Add_2:0", shape=(), dtype=int32)
>>> sess = tf.Session()
>>> print(sess.run(a))
5
```

SESSION的管理



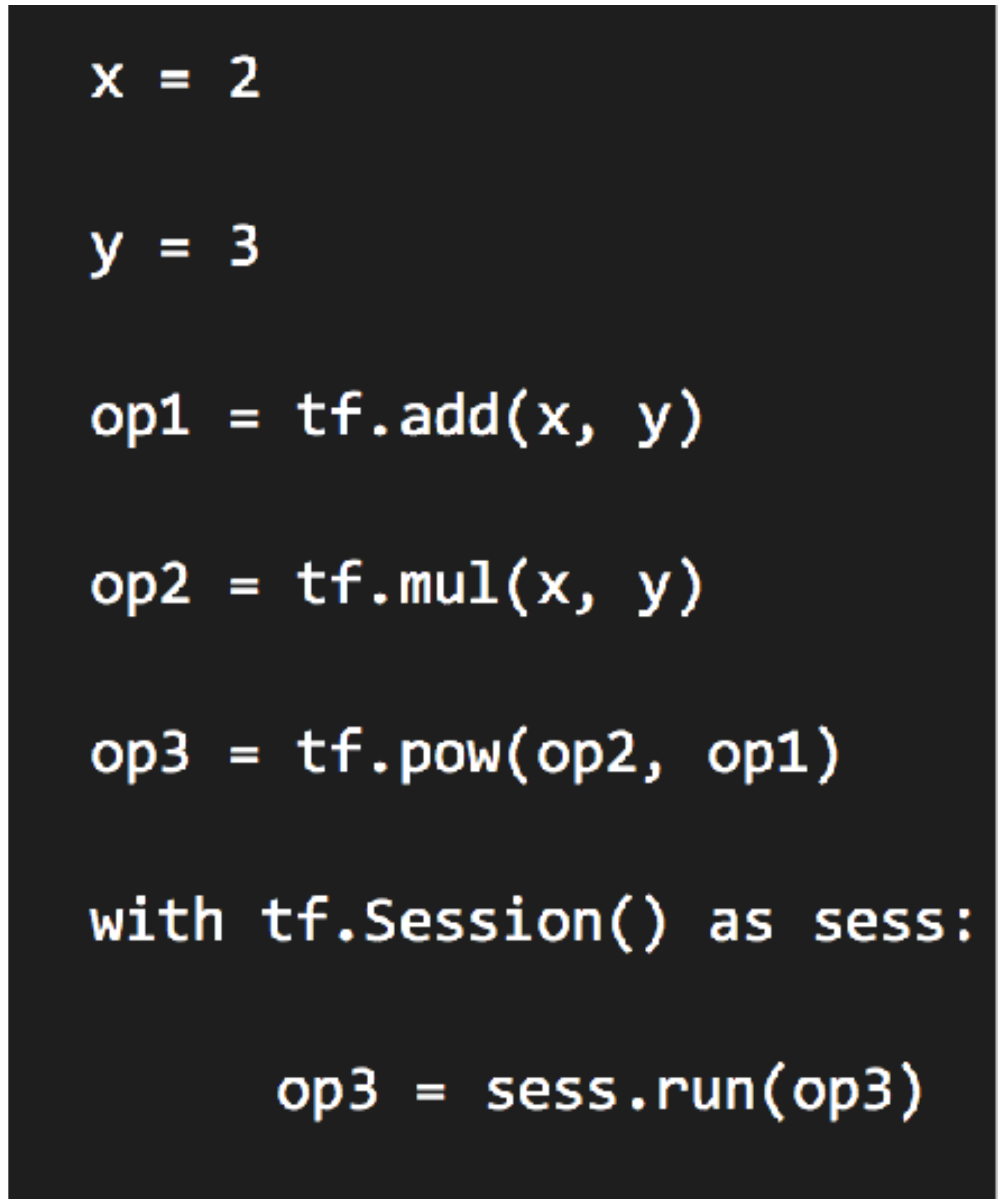
```
IndentationError: unexpected
>>> import tensorflow as tf
>>> a = tf.add(2,3)
>>> sess = tf.Session()
>>> print(sess.run(a))
5
>>> sess.close()
>>> 
```

OR

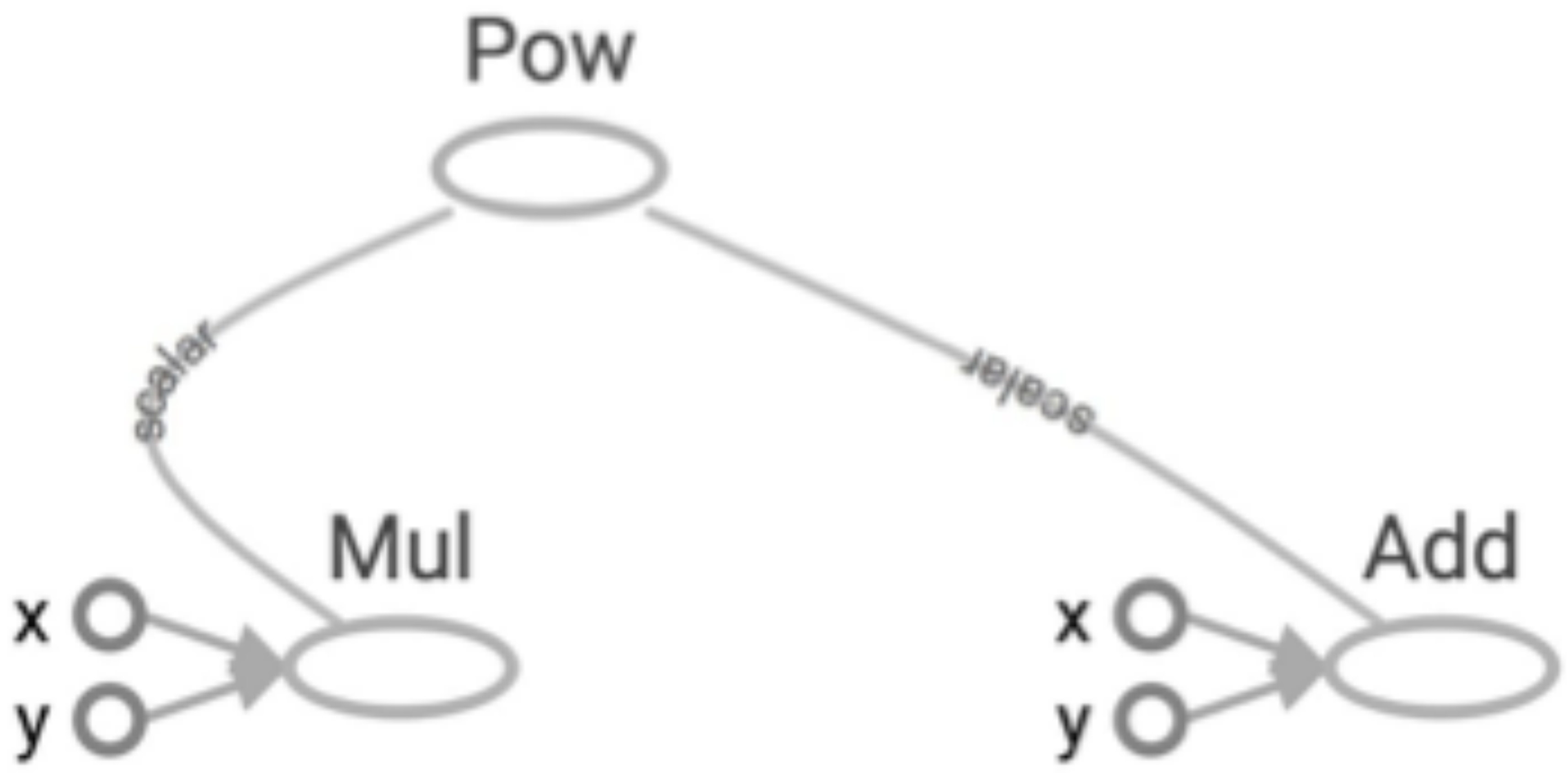


```
>>> sess.close()
>>> with tf.Session() as sess:
...     print(sess.run(a))
...
5
>>> 
```

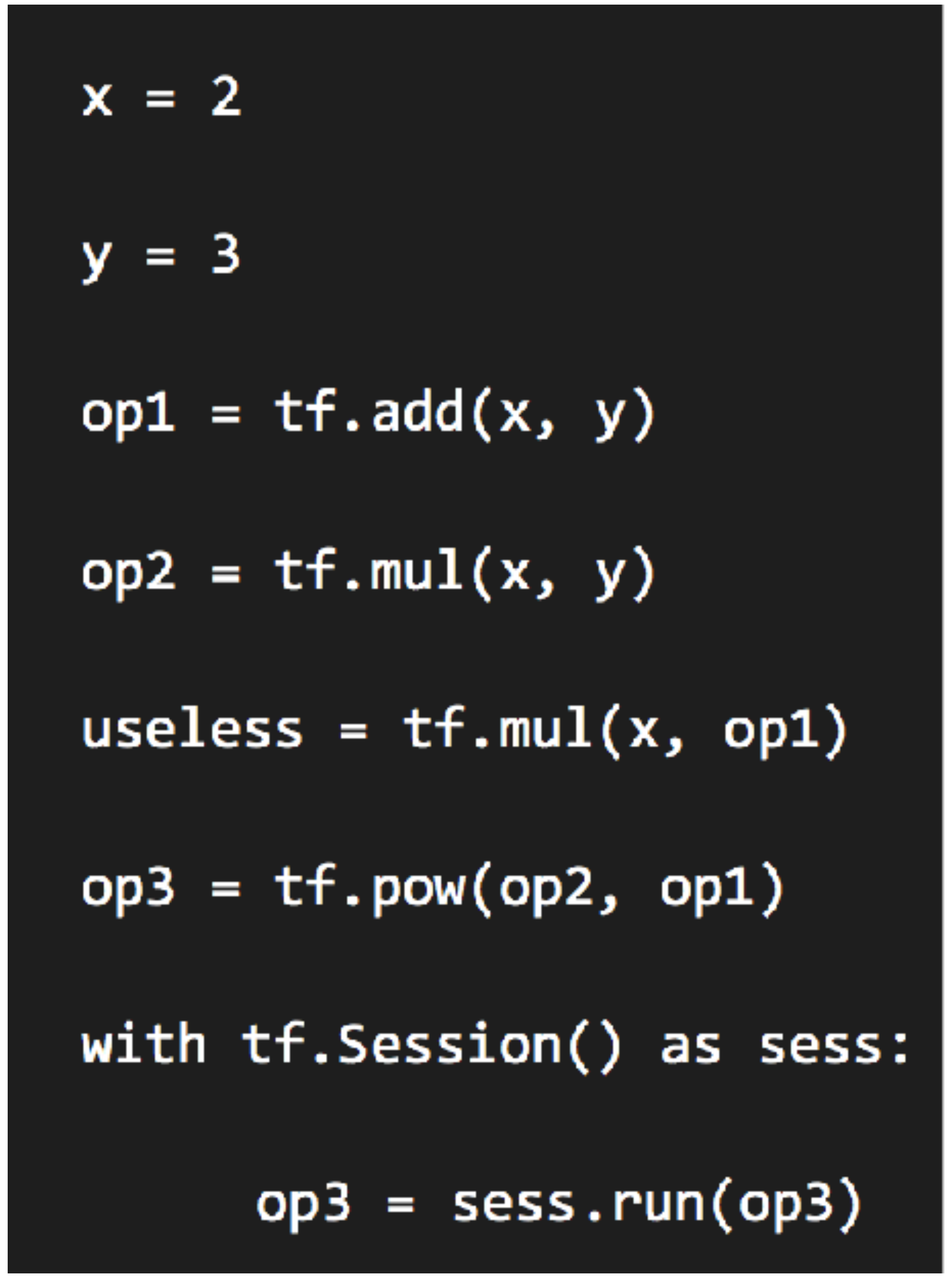
更多的操作



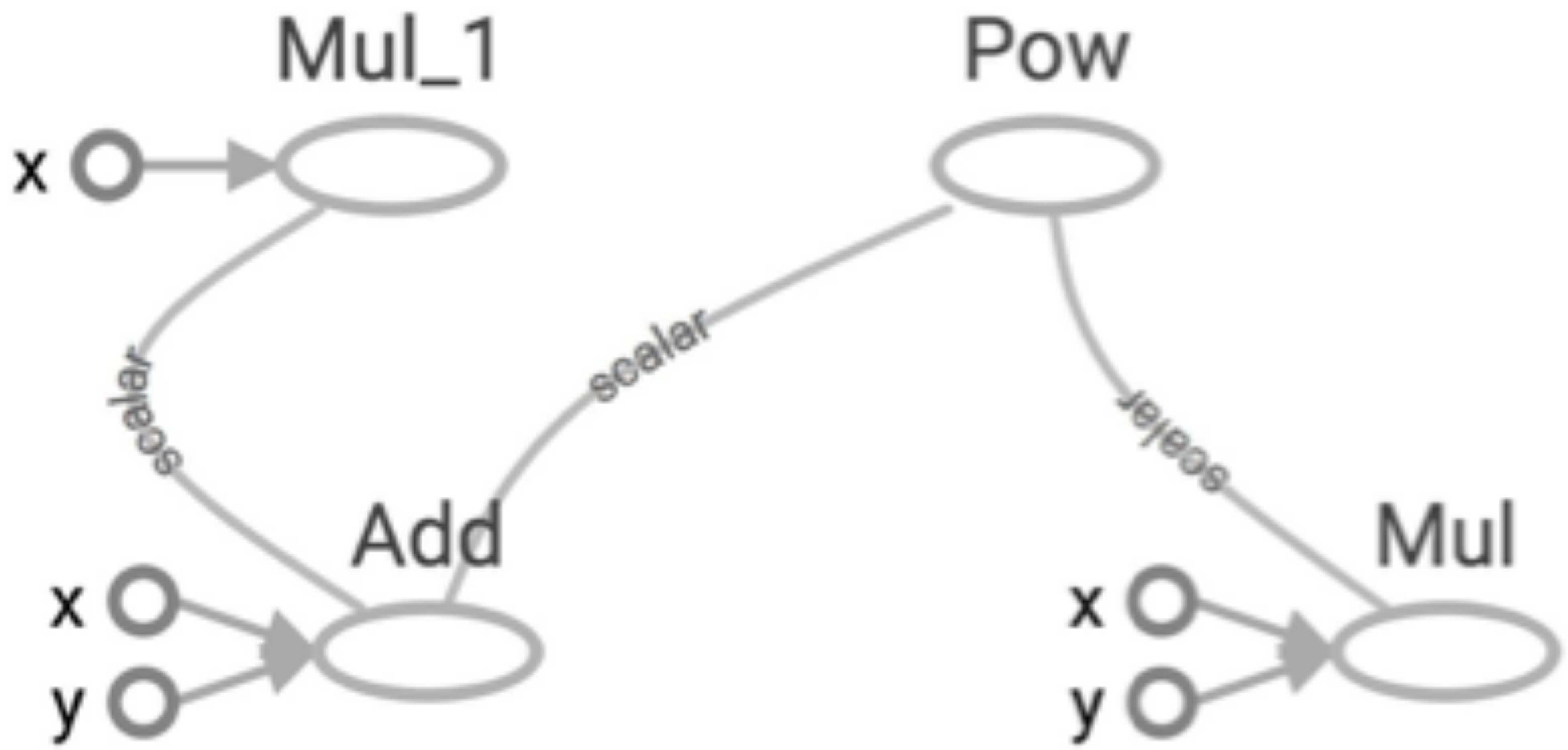
```
x = 2

y = 3

op1 = tf.add(x, y)

op2 = tf.mul(x, y)

op3 = tf.pow(op2, op1)

with tf.Session() as sess:

     op3 = sess.run(op3)
```



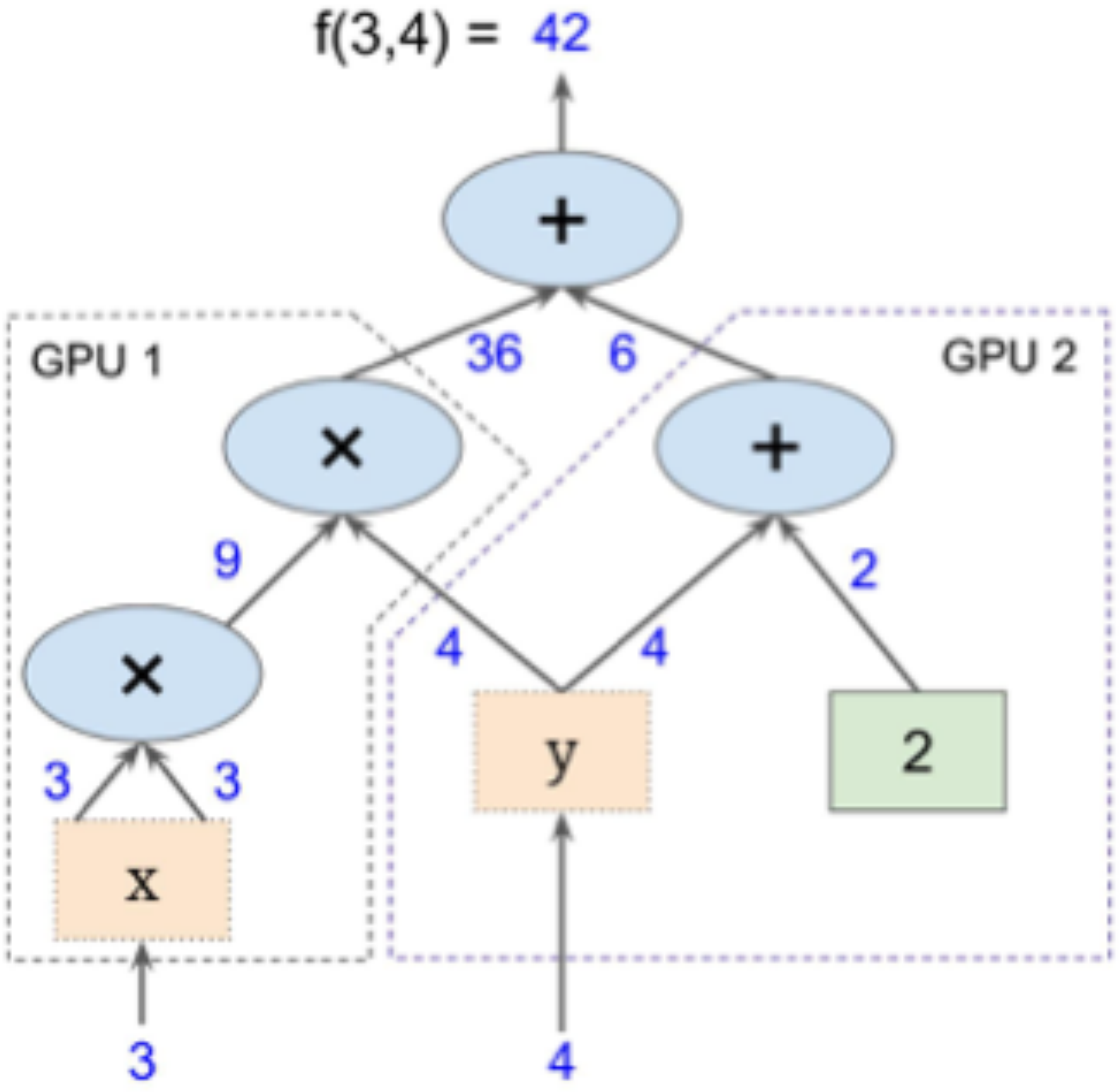
更多操作



```
x = 2

y = 3

op1 = tf.add(x, y)

op2 = tf.mul(x, y)

useless = tf.mul(x, op1)

op3 = tf.pow(op2, op1)

with tf.Session() as sess:

     op3 = sess.run(op3)
```



计算图的好处



计算图可以被分解为不同的部分
然后运行在多个GPU或者CPU之上

构建多个图

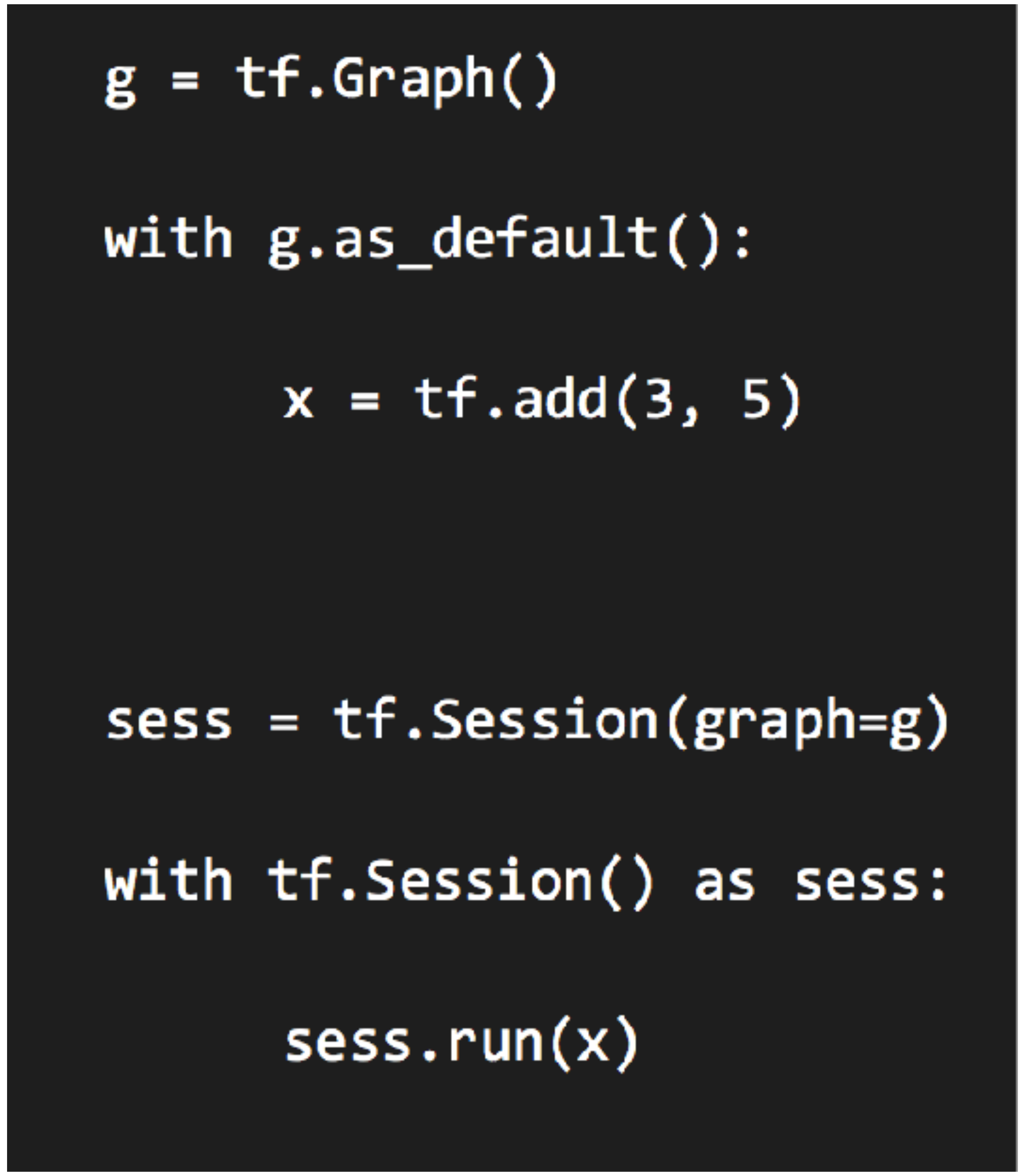
并不推荐构建多个图，因为有以下几点问题：

- 多个图需要多个Session，都占用默认的资源
- 只能通过python/numpy的方式传递数据，丧失了分布式的好处

创建一个图

```
g = tf.Graph()
```

使用图

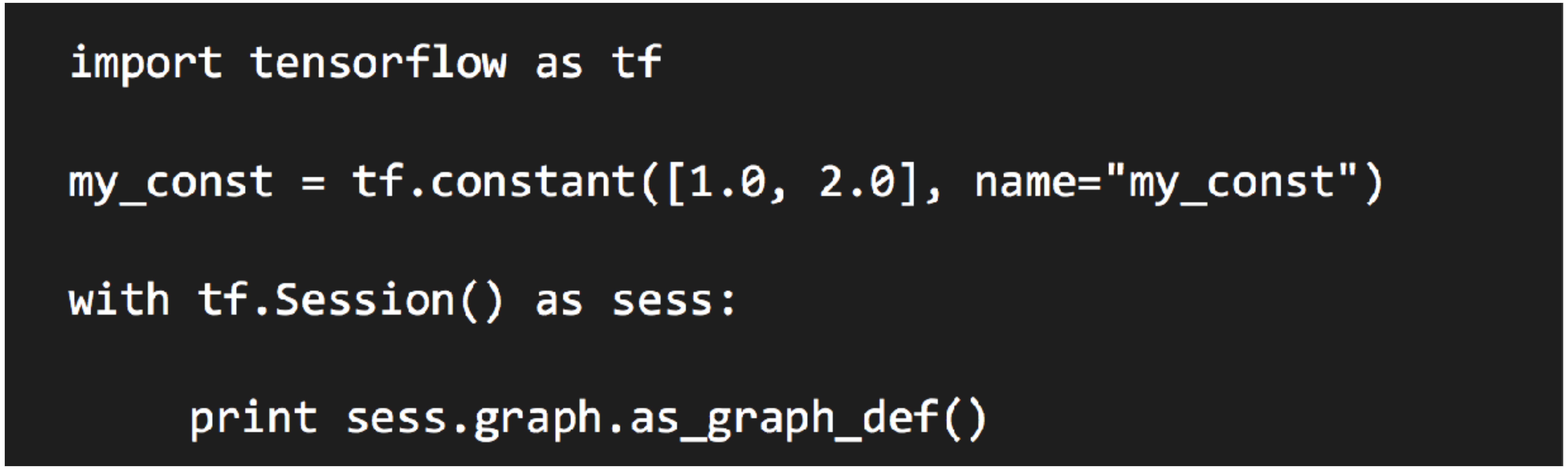


```
g = tf.Graph()
with g.as_default():
    x = tf.add(3, 5)

sess = tf.Session(graph=g)
with tf.Session() as sess:
    sess.run(x)
```

获取默认的图

```
g = tf.get_default_graph()
```

查看图定义



```
import tensorflow as tf

my_const = tf.constant([1.0, 2.0], name="my_const")

with tf.Session() as sess:

    print sess.graph.as_graph_def()
```

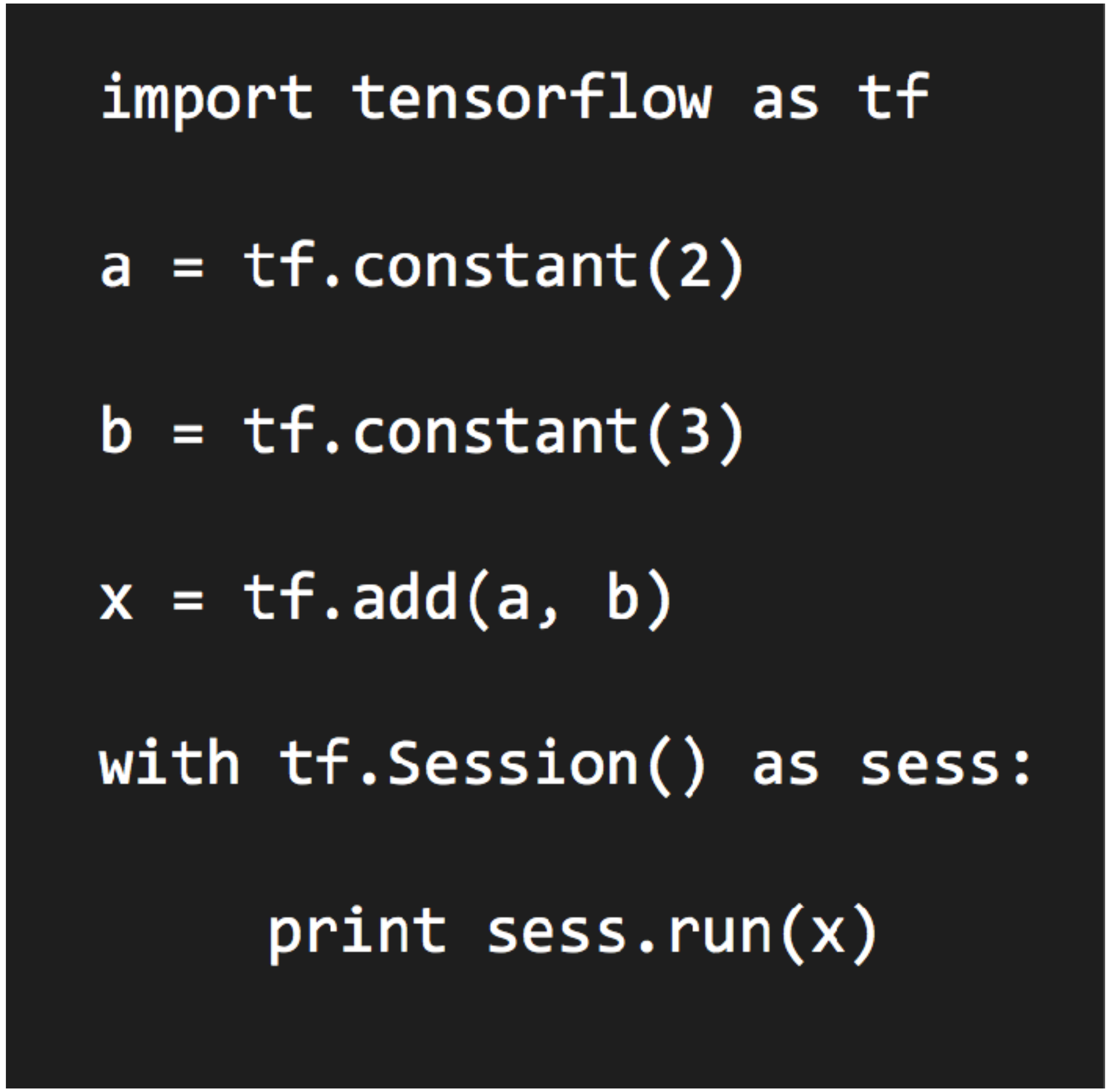
ThoughtWorks®

大数据&人工智能

TENSORFLOW基本操作

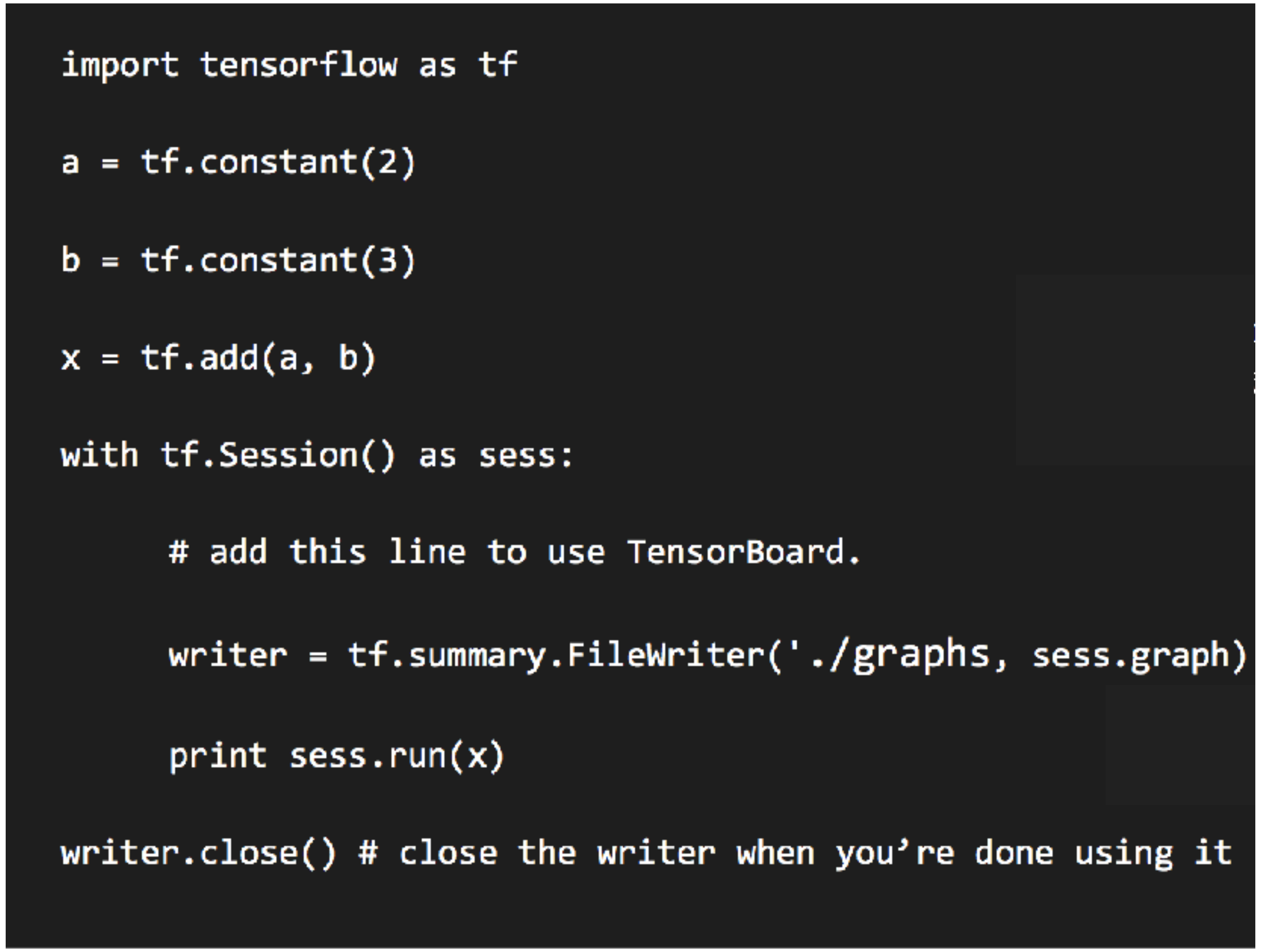
慕课学院
www.mooc.ai

第一个TENSORFLOW程序



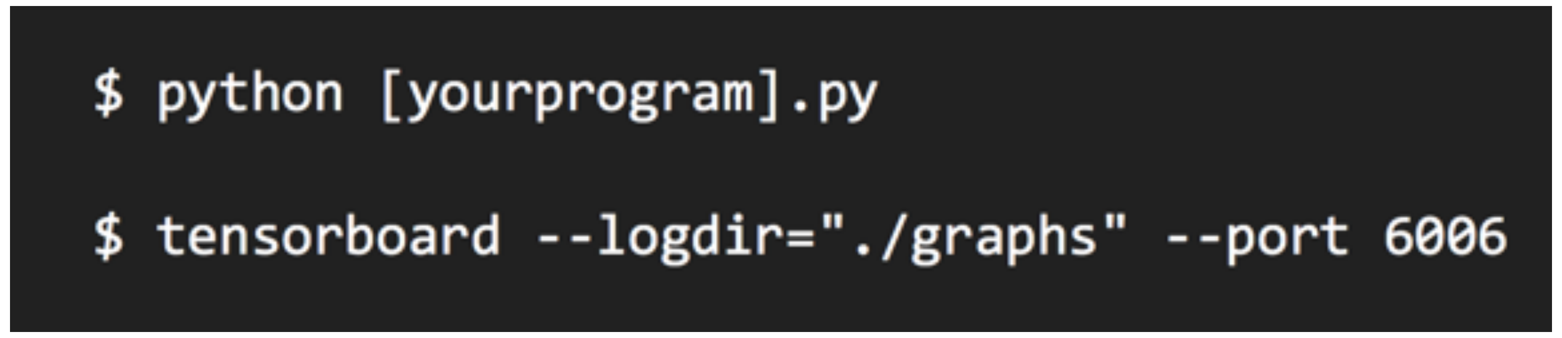
```
import tensorflow as tf

a = tf.constant(2)

b = tf.constant(3)

x = tf.add(a, b)

with tf.Session() as sess:

    print sess.run(x)
```

TENSOR BOARD



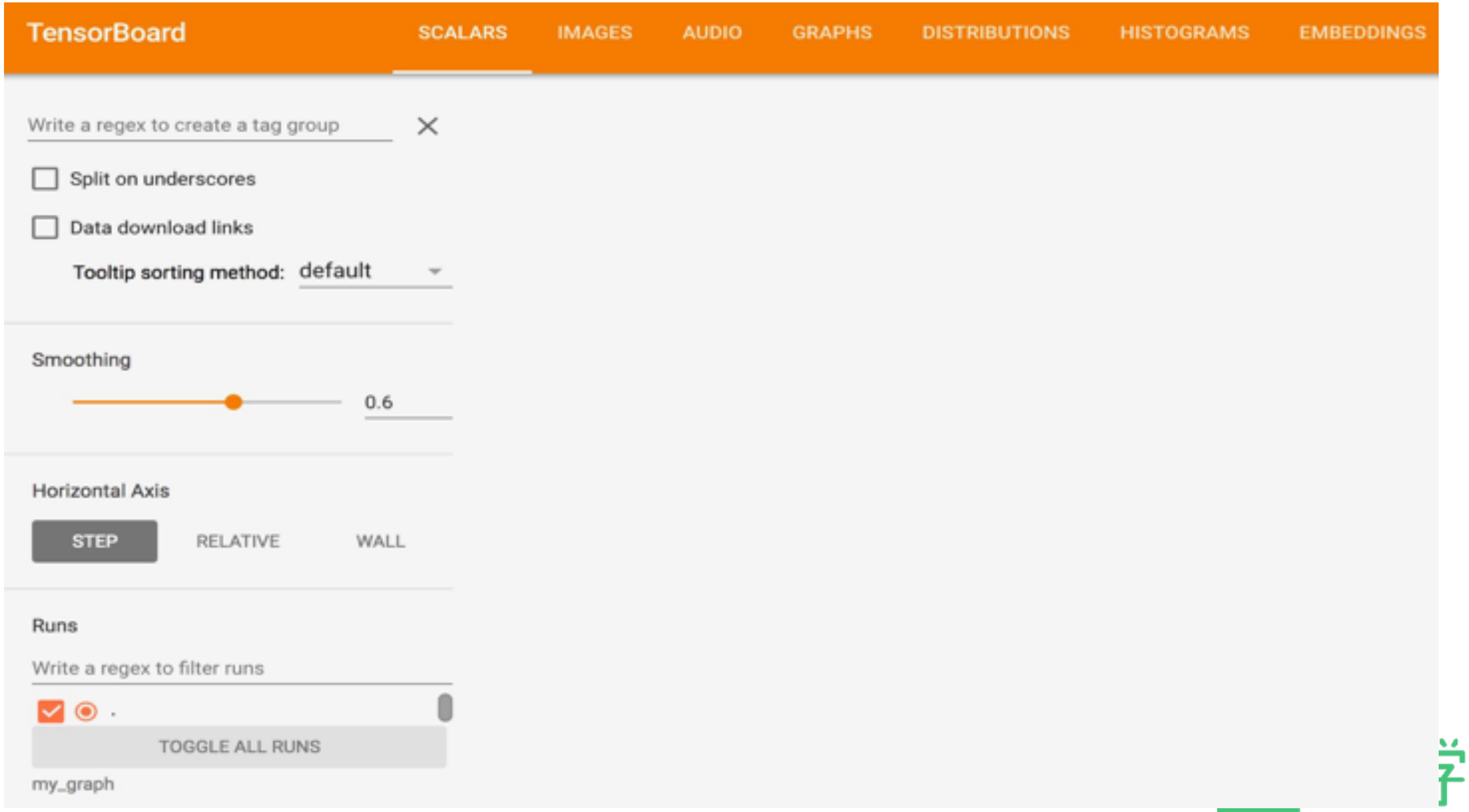
```
import tensorflow as tf

a = tf.constant(2)

b = tf.constant(3)

x = tf.add(a, b)

with tf.Session() as sess:

      # add this line to use TensorBoard.

      writer = tf.summary.FileWriter('./graphs, sess.graph)

      print sess.run(x)

writer.close() # close the writer when you’re done using it
```

可以通过6006端口访问



```
$ python [yourprogram].py

$ tensorboard --logdir="./graphs" --port 6006
```

TENSOR BOARD



Write a regex to create a tag group

□ Split on underscores

□ Data download links

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

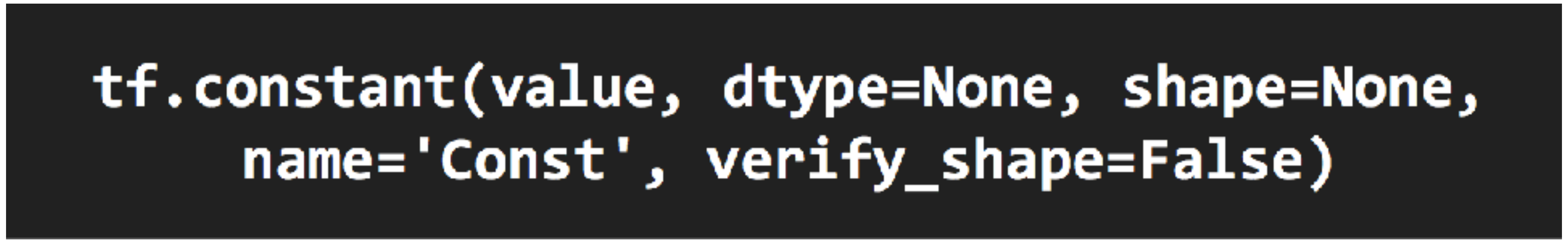
Write a regex to filter runs

☑ .

TOGGLE ALL RUNS

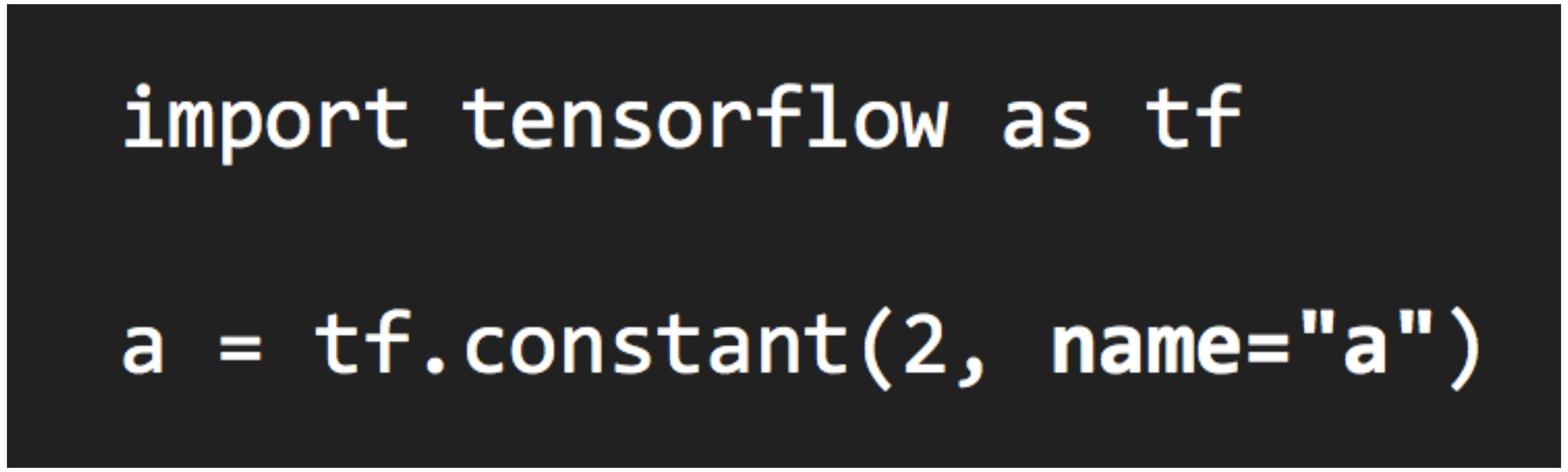
my_graph

创建常量



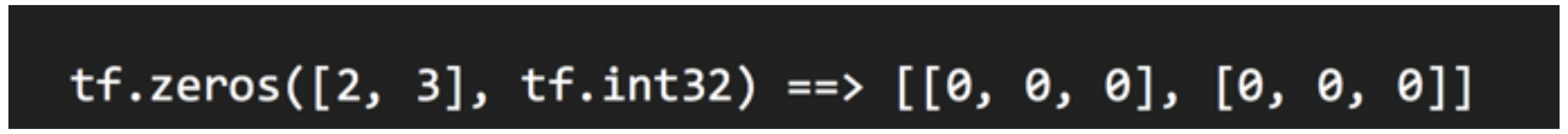
```
tf.constant(value, dtype=None, shape=None,
  name='Const', verify_shape=False)
```

给变量起一个自己的名字

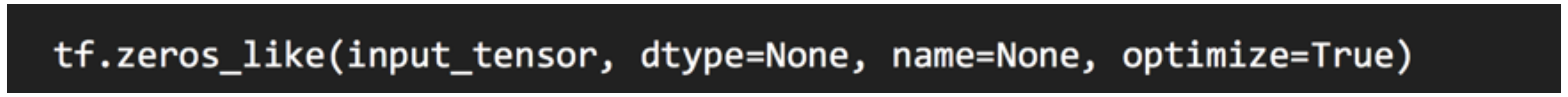


```
import tensorflow as tf

a = tf.constant(2, name="a")
```

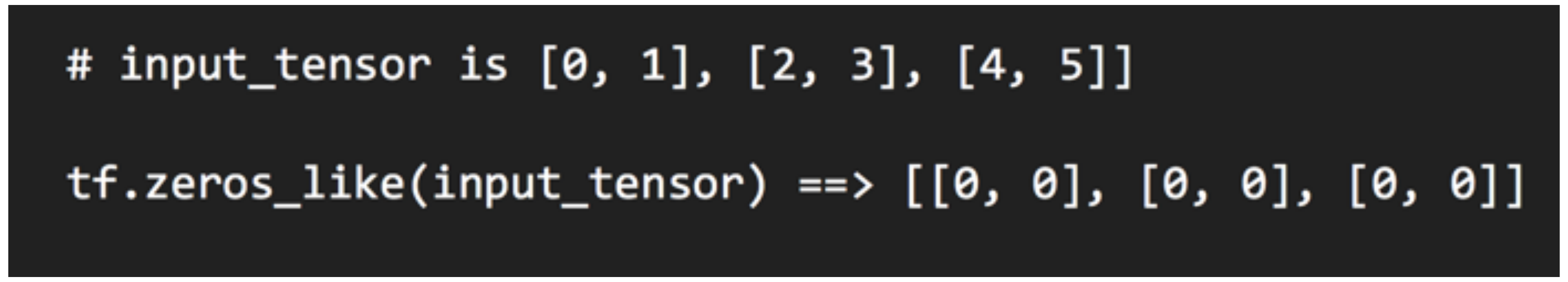
指定特定的初始值



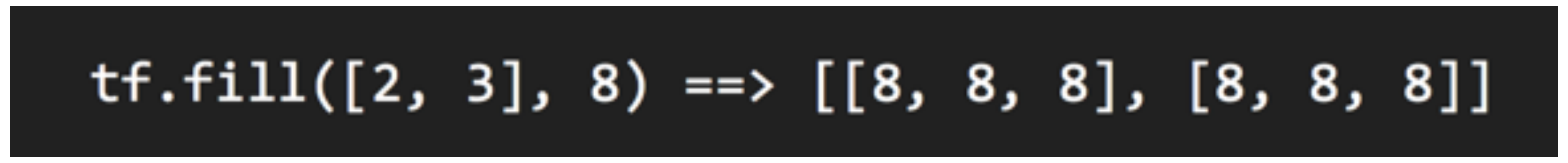
```
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
```



```
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True)
```

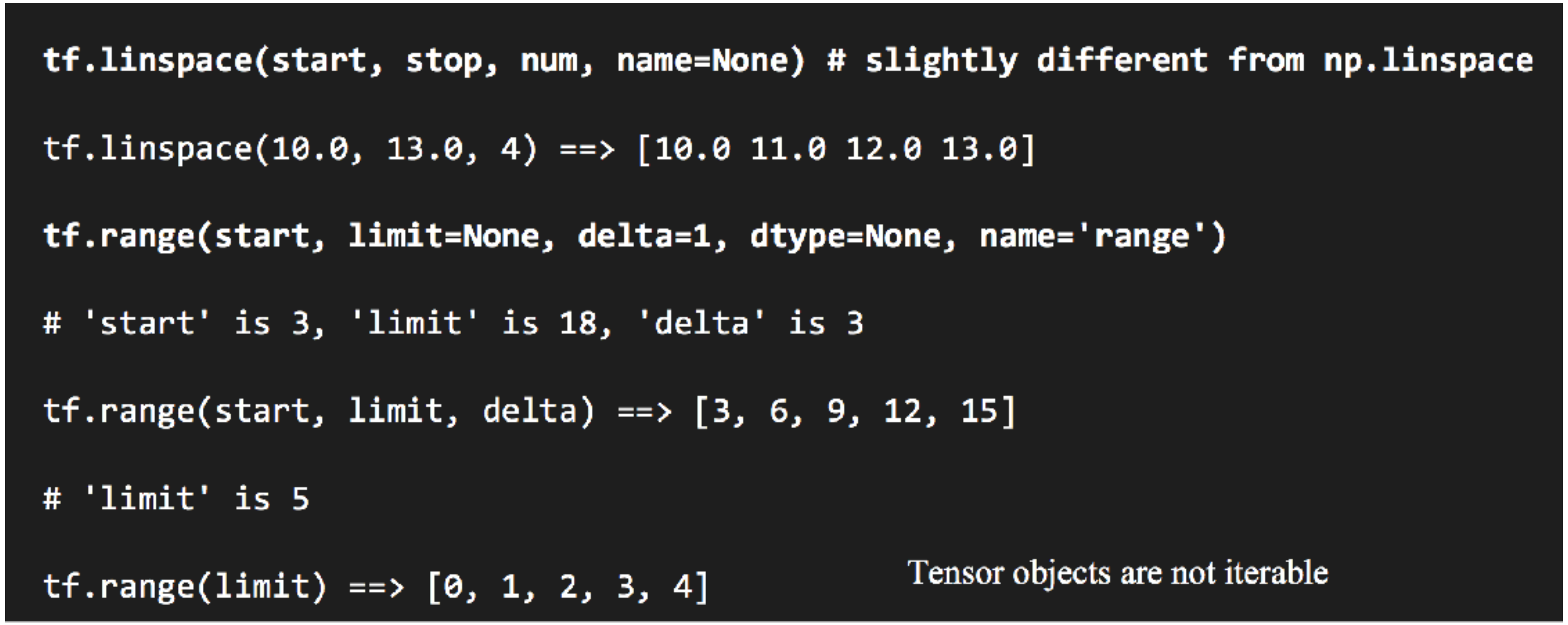


```
# input_tensor is [0, 1], [2, 3], [4, 5]]

tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]
```



```
tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]
```

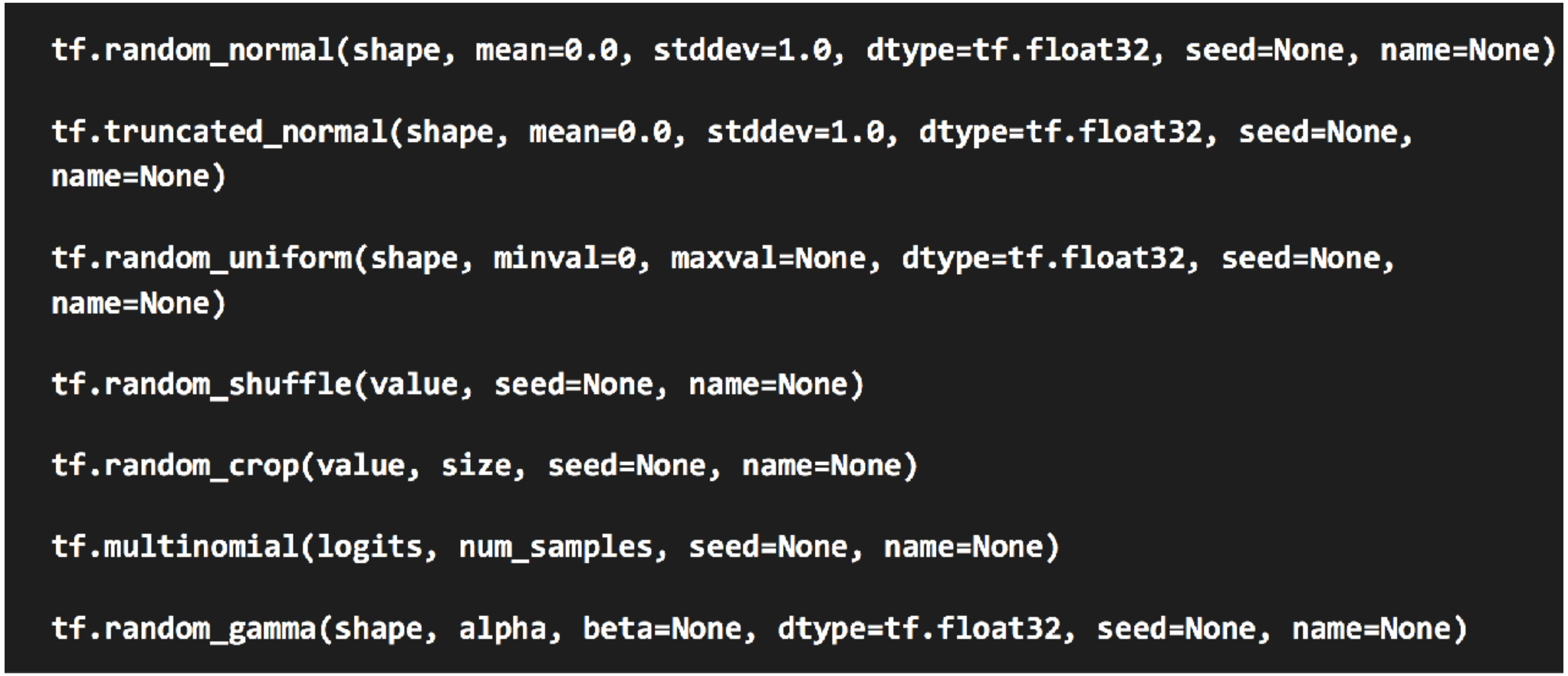
指定特定的初始值



```
tf.linspace(start, stop, num, name=None) # slightly different from np.linspace

tf.linspace(10.0, 13.0, 4) ==> [10.0 11.0 12.0 13.0]

tf.range(start, limit=None, delta=1, dtype=None, name='range')

# 'start' is 3, 'limit' is 18, 'delta' is 3

tf.range(start, limit, delta) ==> [3, 6, 9, 12, 15]

# 'limit' is 5

tf.range(limit) ==> [0, 1, 2, 3, 4]
```

Tensor objects are not iterable

随机数



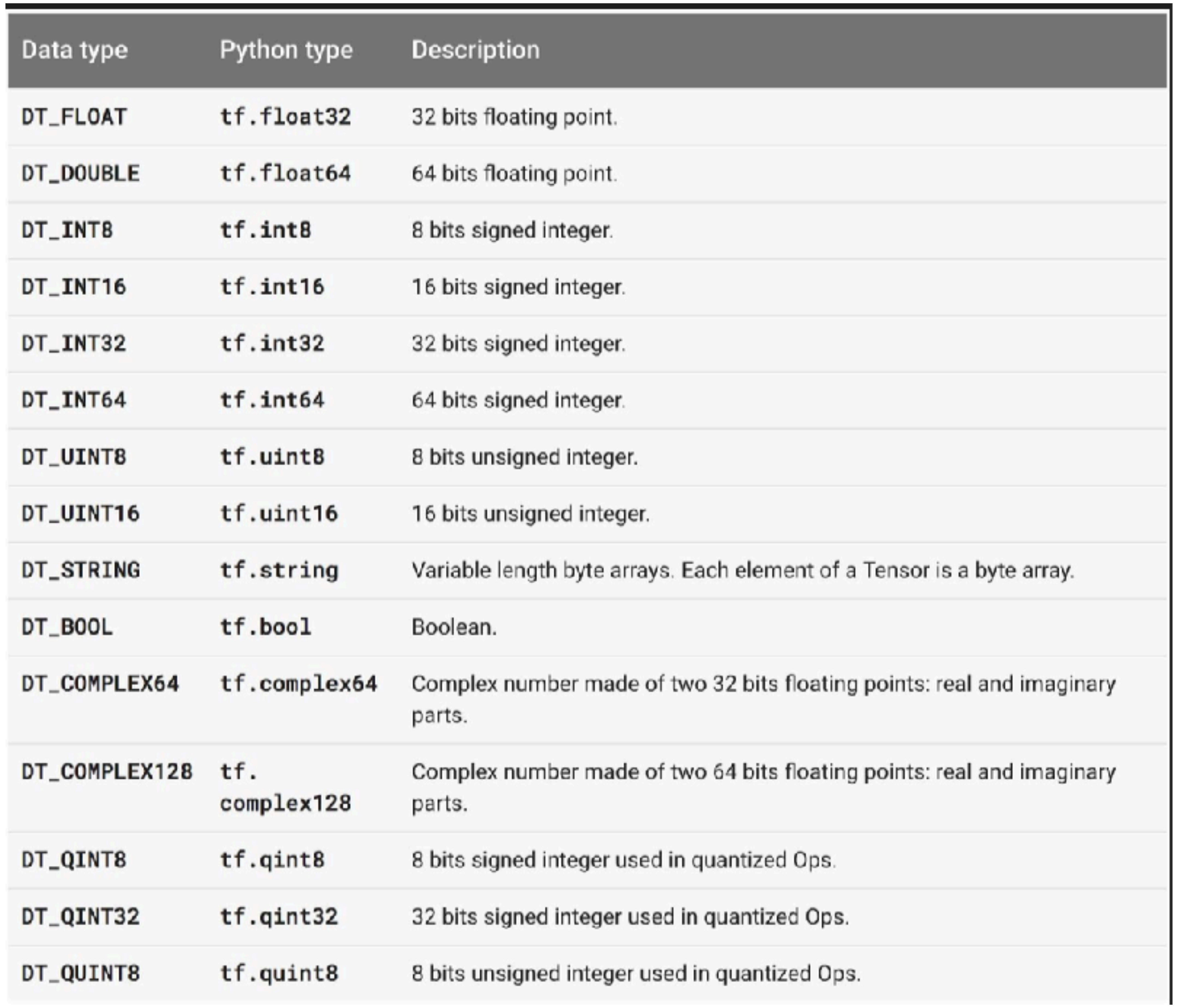
```
tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None,
name=None)

tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None,
name=None)

tf.random_shuffle(value, seed=None, name=None)

tf.random_crop(value, size, seed=None, name=None)

tf.multinomial(logits, num_samples, seed=None, name=None)

tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)
```

ThoughtWorks®

大数据&人工智能

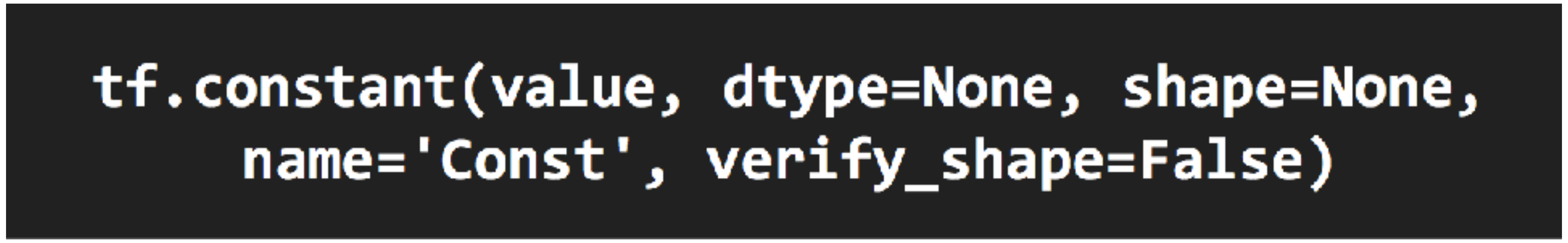
TENSOR类型

TF数据类型



Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

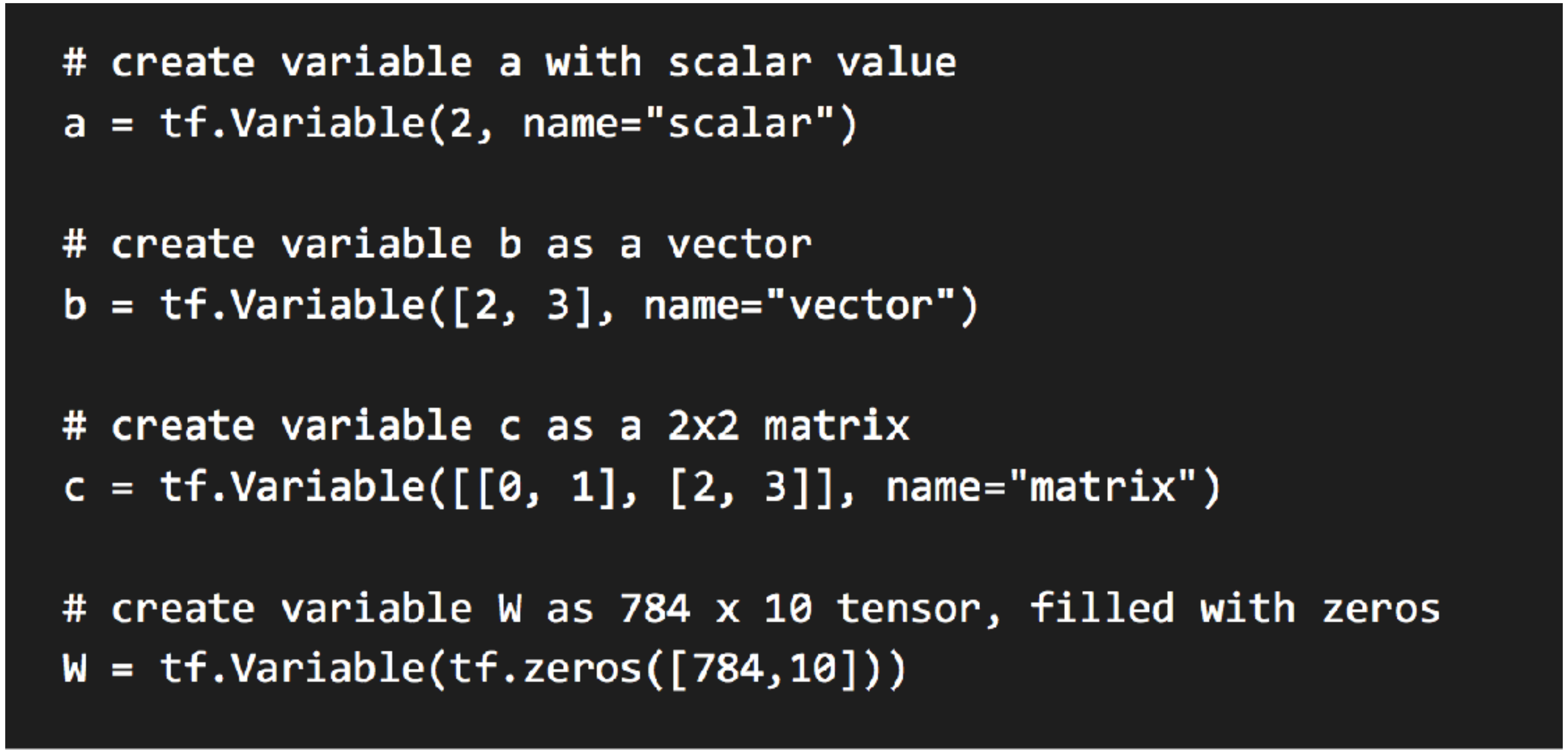
常量



```
tf.constant(value, dtype=None, shape=None,
    name='Const', verify_shape=False)
```

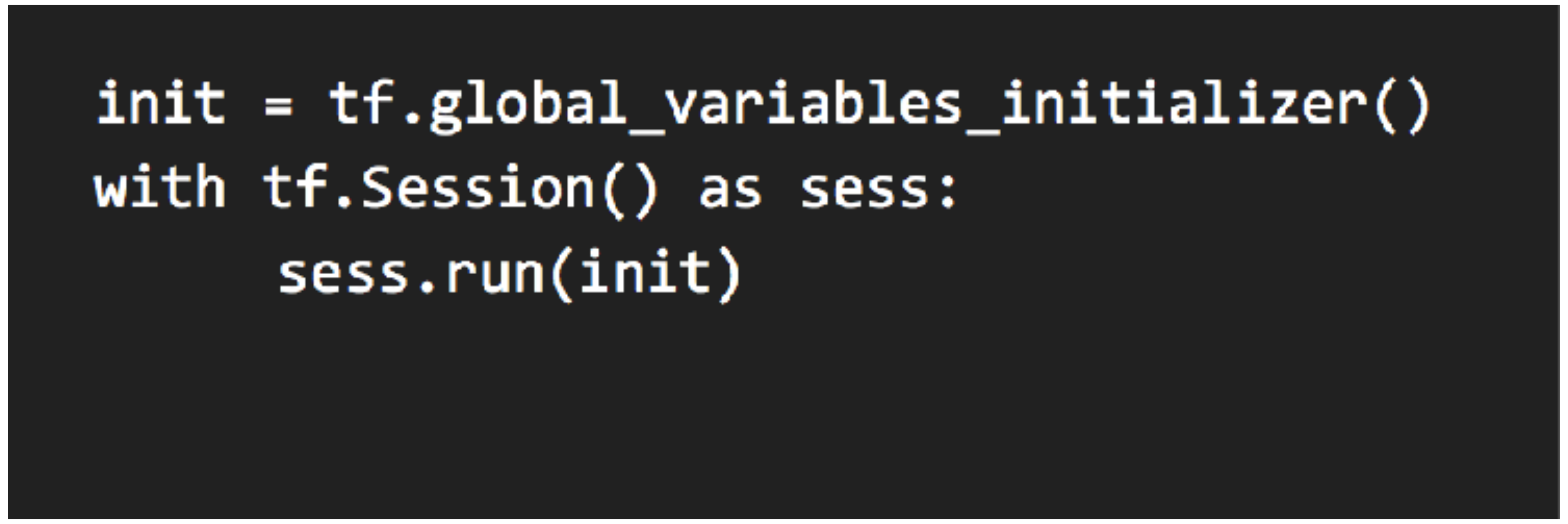
- 常量占据的空间在定义的时候发生
- 当常量的内容特别大的时候，非常的浪费资源
- 如果不是特别需要，不建议使用常量，尽量以变量代替

变量

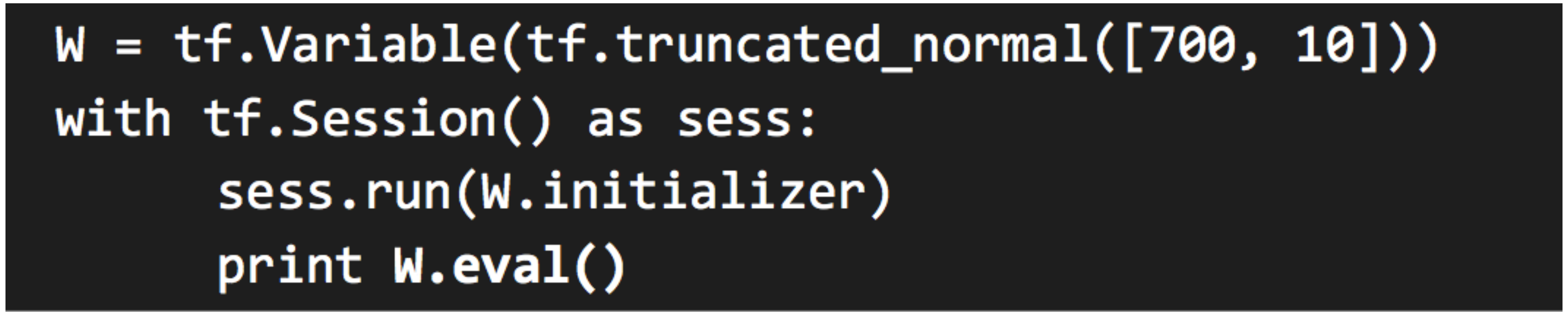


```
# create variable a with scalar value
a = tf.Variable(2, name="scalar")

# create variable b as a vector
b = tf.Variable([2, 3], name="vector")

# create variable c as a 2x2 matrix
c = tf.Variable([[0, 1], [2, 3]], name="matrix")

# create variable W as 784 x 10 tensor, filled with zeros
W = tf.Variable(tf.zeros([784,10]))
```

变量必须被初始化



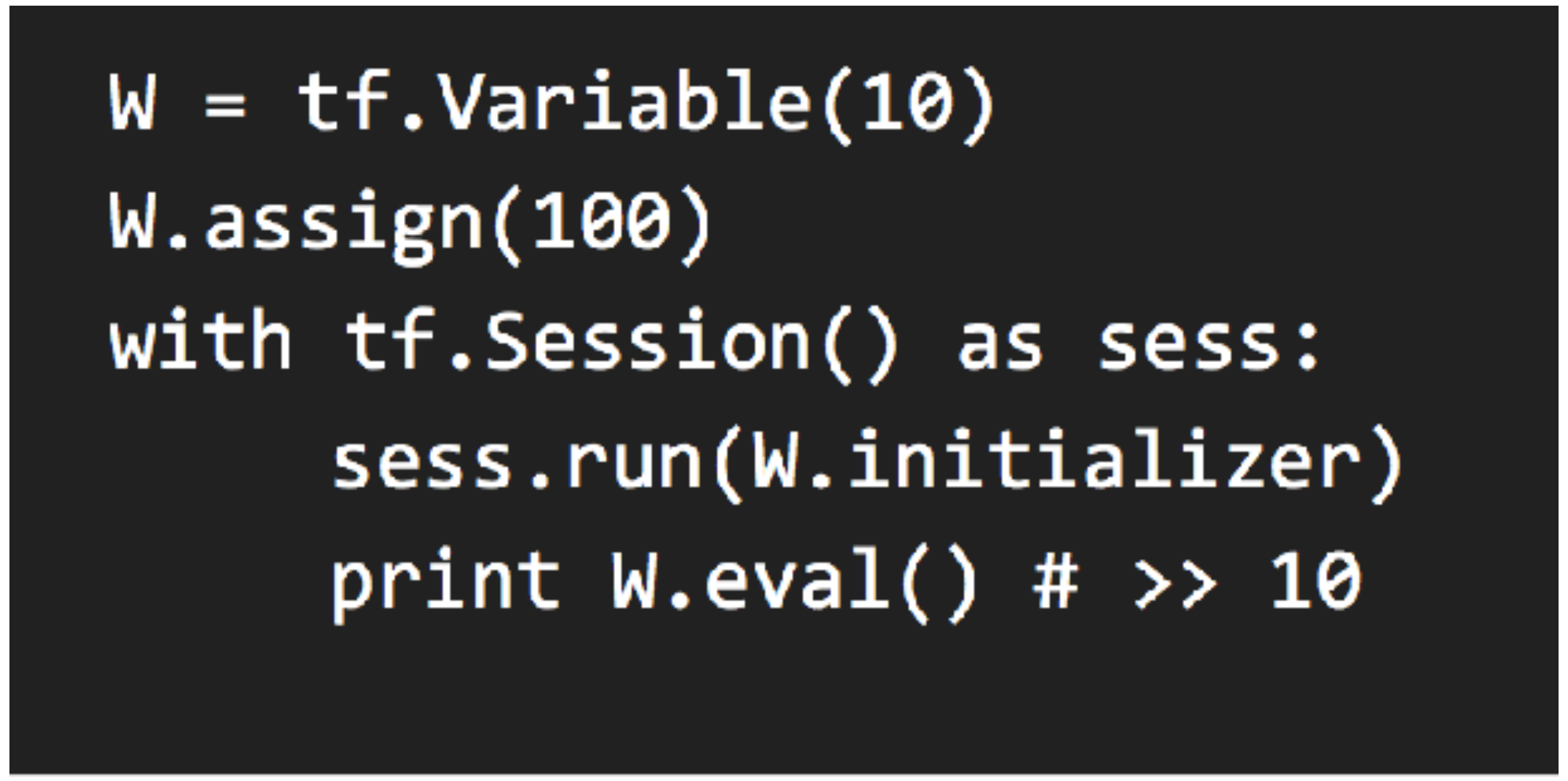
```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
```



```
W = tf.Variable(tf.truncated_normal([700, 10]))
with tf.Session() as sess:
    sess.run(W.initializer)
    print W.eval()
```

ThoughtWorks®

变量赋值

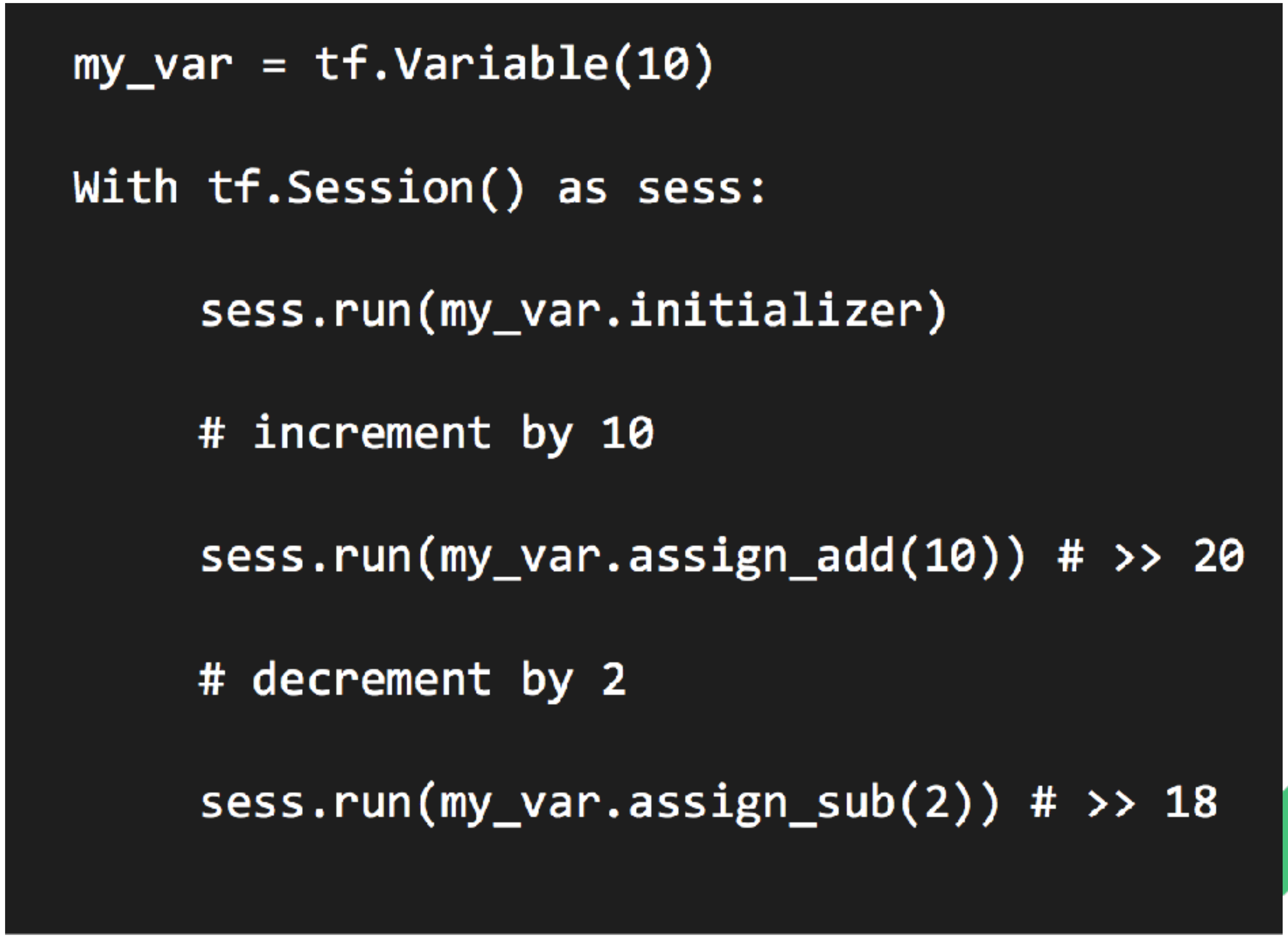


```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> 10
```

为什么输出是10?

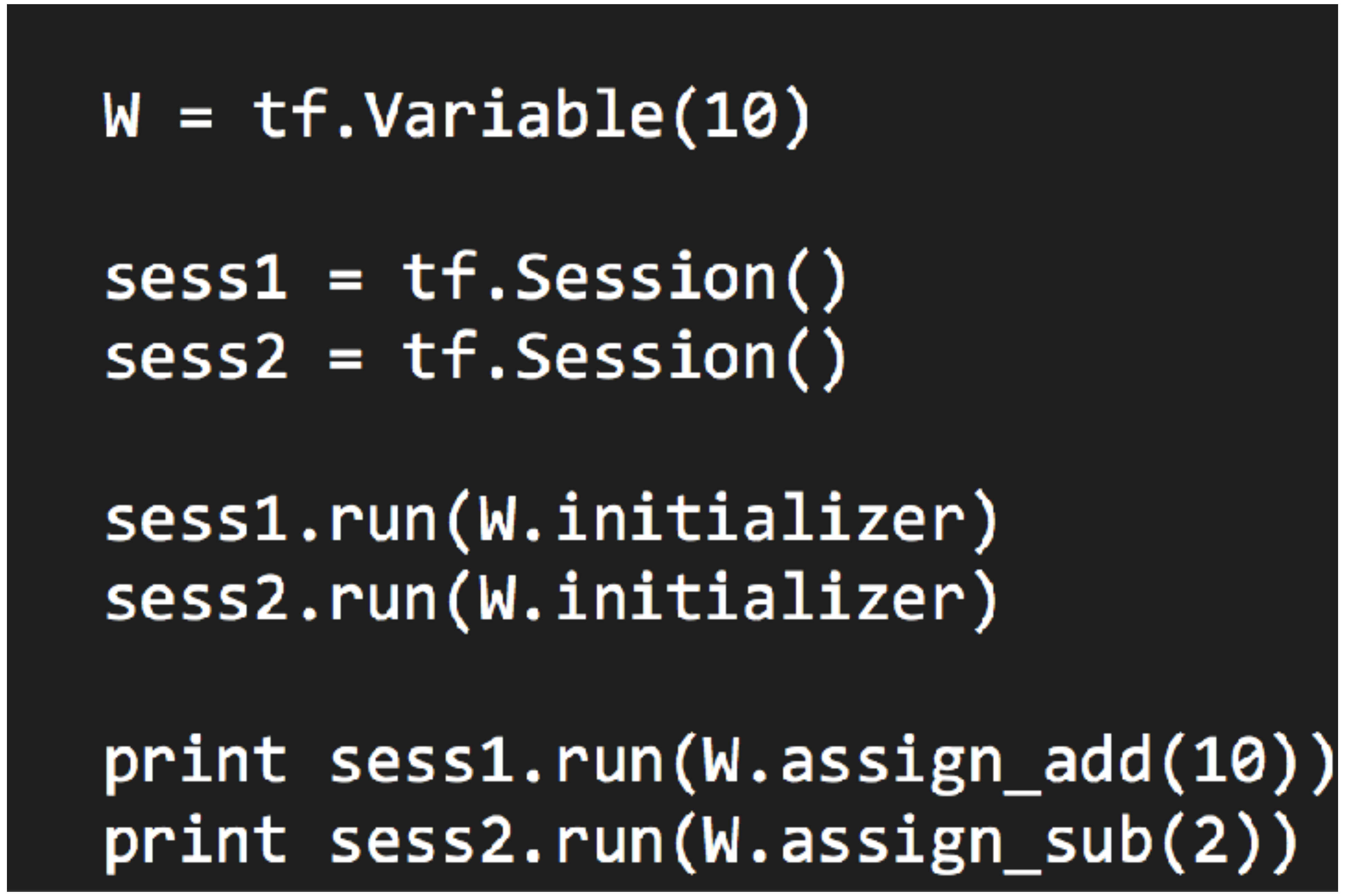
变量赋值

assign_add() and assign_sub()



```
my_var = tf.Variable(10)

With tf.Session() as sess:

    sess.run(my_var.initializer)

    # increment by 10

    sess.run(my_var.assign_add(10)) # >> 20

    # decrement by 2

    sess.run(my_var.assign_sub(2)) # >> 18
```

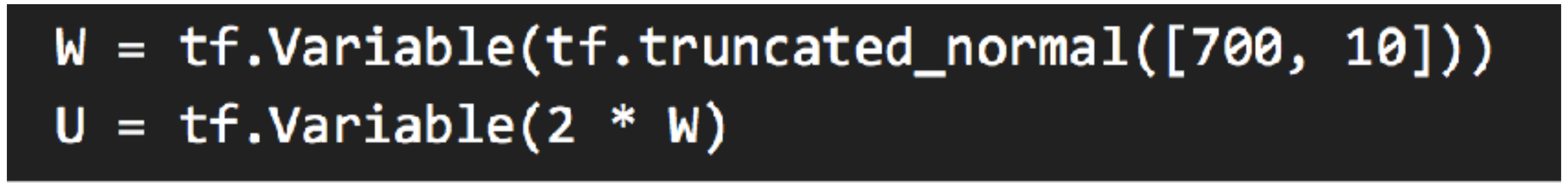
多个SESSION之间数据不共享



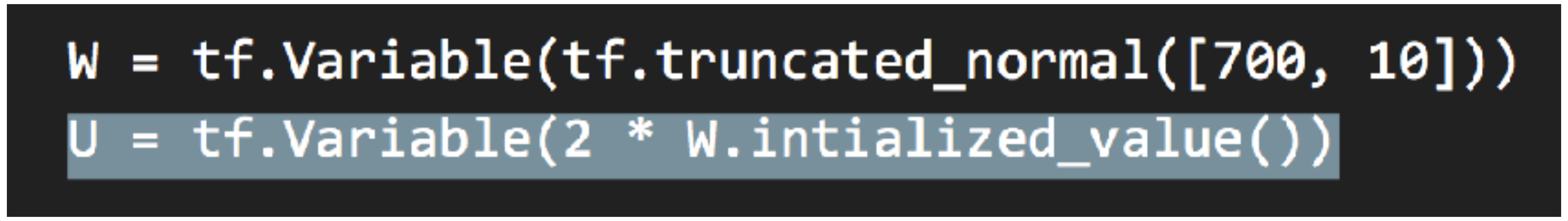
```
W = tf.Variable(10)

sess1 = tf.Session()
sess2 = tf.Session()

sess1.run(W.initializer)
sess2.run(W.initializer)

print sess1.run(W.assign_add(10))
print sess2.run(W.assign_sub(2))
```

```
20
8
```

变量依赖

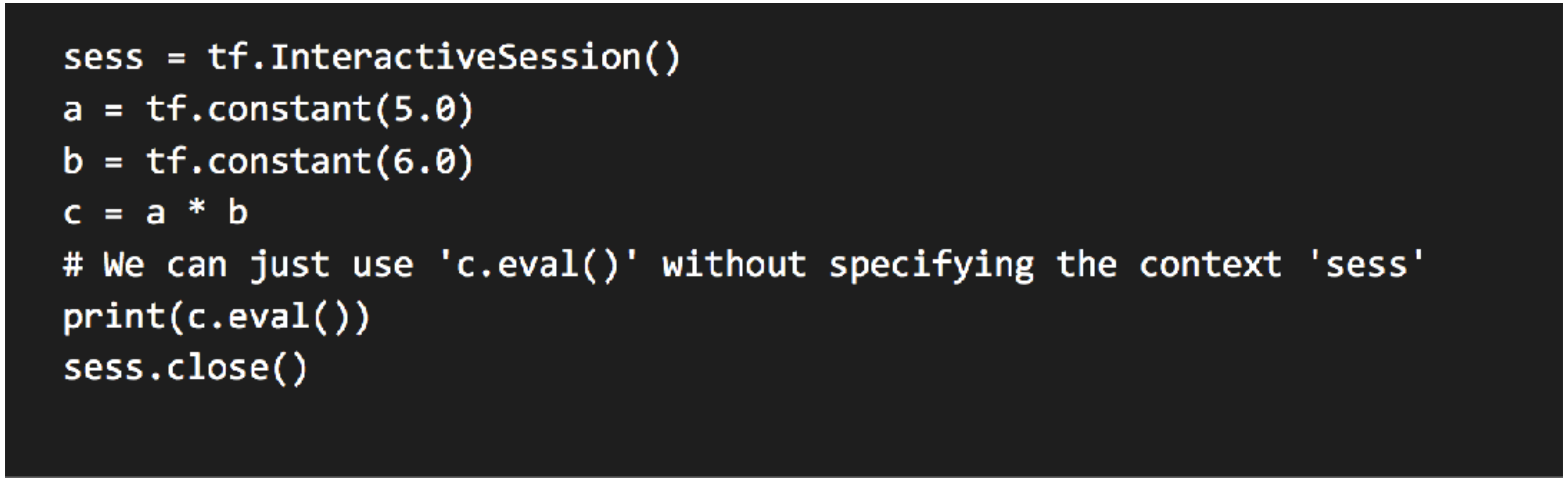


```
W = tf.Variable(tf.truncated_normal([700, 10]))
U = tf.Variable(2 * W)
```



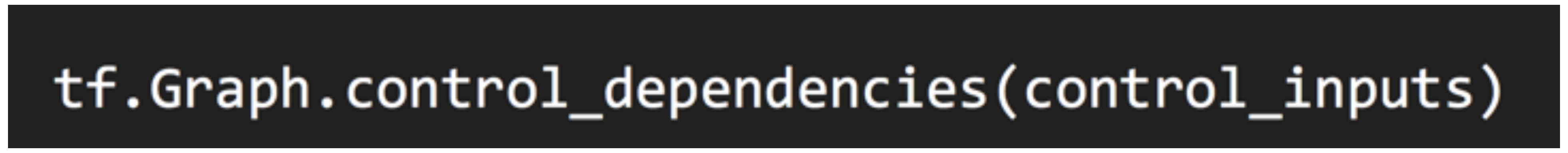
```
W = tf.Variable(tf.truncated_normal([700, 10]))
U = tf.Variable(2 * W.intialized_value())
```

交互式SESSION

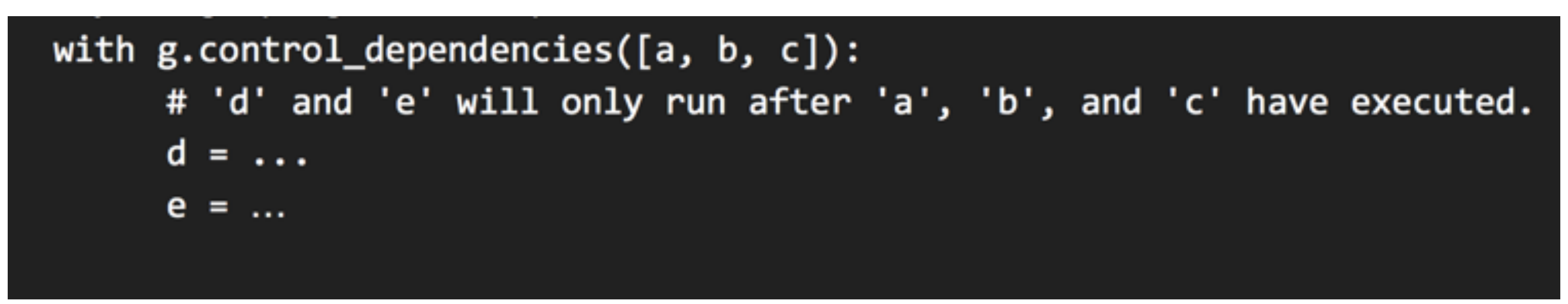


```
sess = tf.InteractiveSession()
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
# We can just use 'c.eval()' without specifying the context 'sess'
print(c.eval())
sess.close()
```

依赖



```
tf.Graph.control_dependencies(control_inputs)
```



```
with g.control_dependencies([a, b, c]):
     # 'd' and 'e' will only run after 'a', 'b', and 'c' have executed.
     d = ...
     e = ...
```

ThoughtWorks®

大数据&人工智能

占位符和输入

回顾一下

一个TF的程序有两个阶段：

- 定义图
- 在一个session里面执行创建的op

占位符

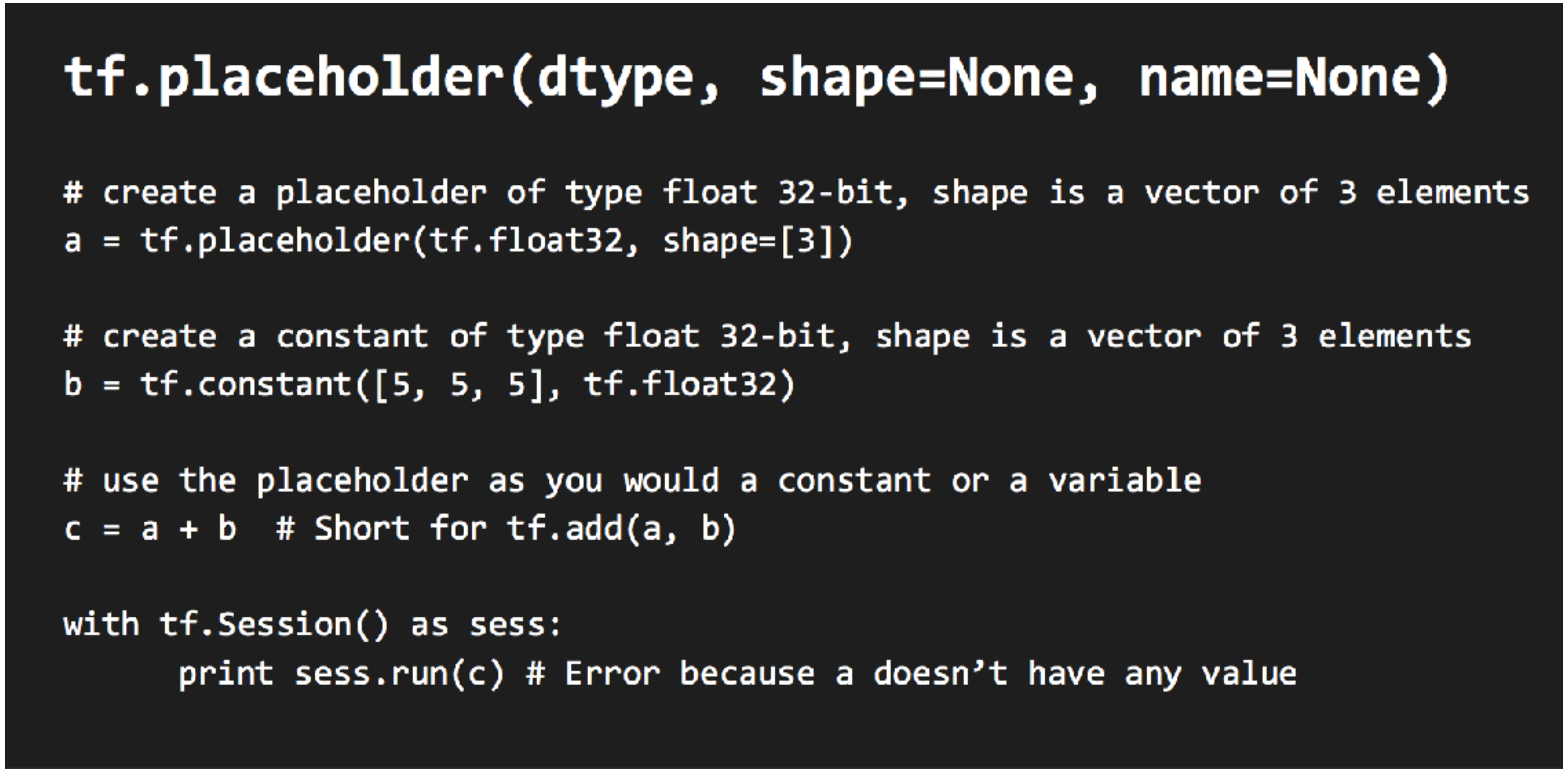
我们在创建图的时候，不必知道tensor的内容是什么
因为它仅仅是计算的时候才需要

占位符

$$f(x, y) = x*2 + y$$

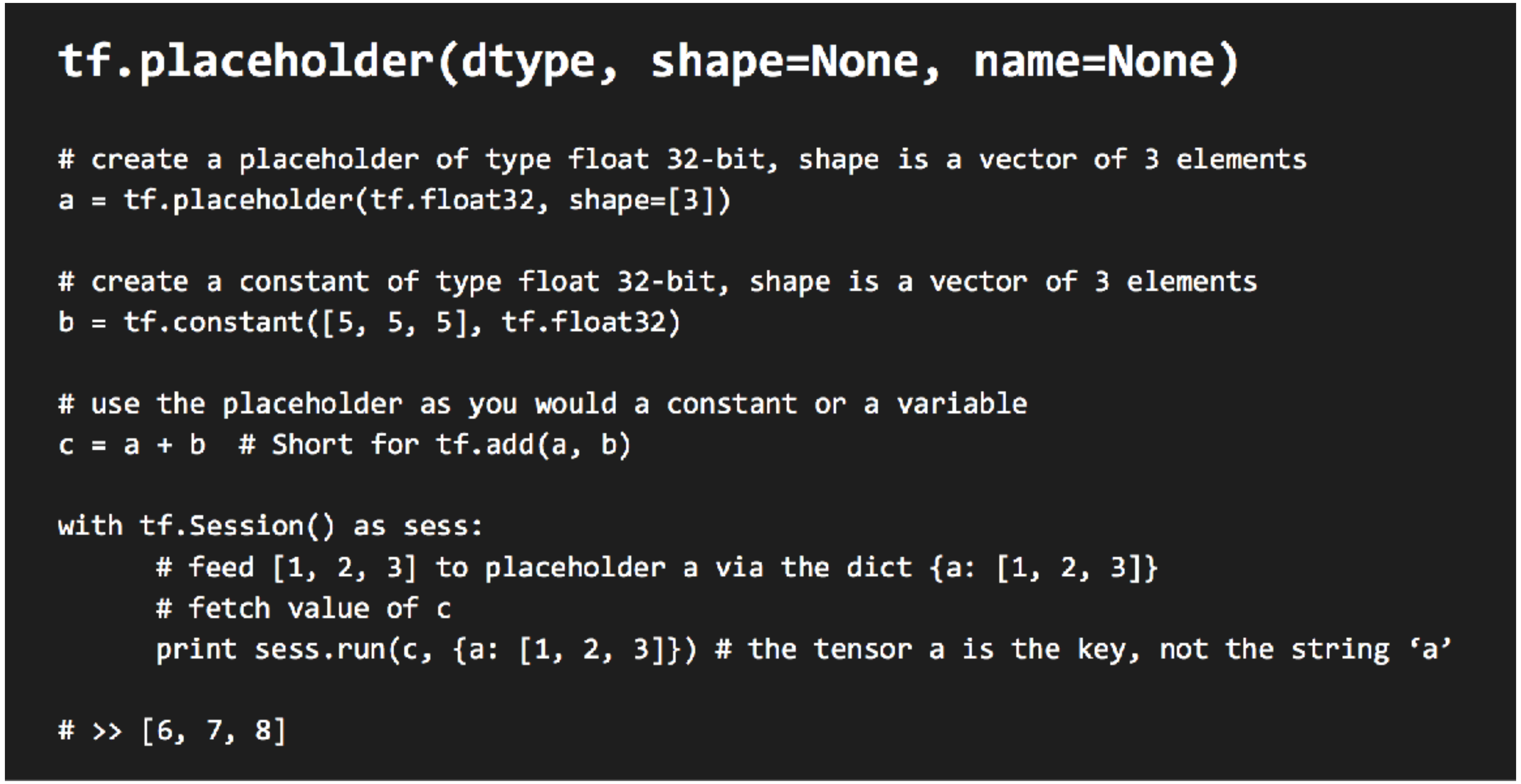
我们可以定义这样一个公式而不必知道x，y的值是什么

占位符



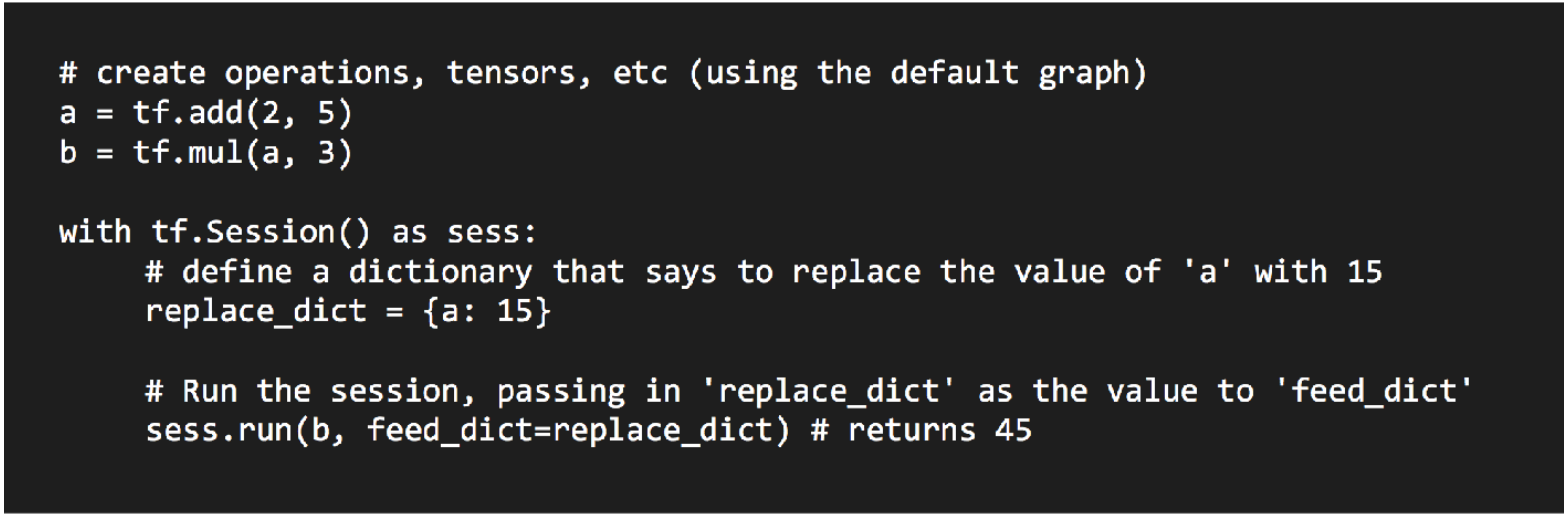
```
tf.placeholder(dtype, shape=None, name=None)

# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

# create a constant of type float 32-bit, shape is a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # Short for tf.add(a, b)

with tf.Session() as sess:
      print sess.run(c) # Error because a doesn’t have any value
```

使用字典作为输入



```
tf.placeholder(dtype, shape=None, name=None)

# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

# create a constant of type float 32-bit, shape is a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # Short for tf.add(a, b)

with tf.Session() as sess:
      # feed [1, 2, 3] to placeholder a via the dict {a: [1, 2, 3]}
      # fetch value of c
      print sess.run(c, {a: [1, 2, 3]}) # the tensor a is the key, not the string ‘a’

# >> [6, 7, 8]
```

使用字典作为输入



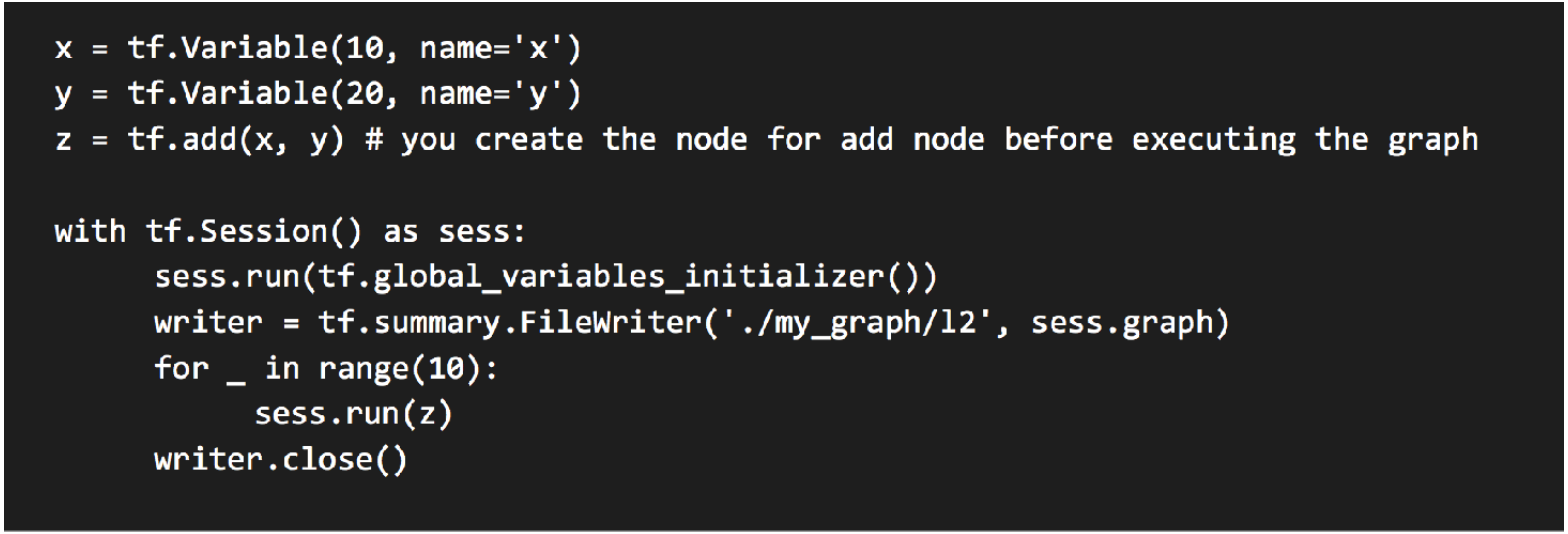
```
# create operations, tensors, etc (using the default graph)
a = tf.add(2, 5)
b = tf.mul(a, 3)

with tf.Session() as sess:
     # define a dictionary that says to replace the value of 'a' with 15
     replace_dict = {a: 15}

     # Run the session, passing in 'replace_dict' as the value to 'feed_dict'
     sess.run(b, feed_dict=replace_dict) # returns 45
```

ThoughtWorks®

大数据&人工智能

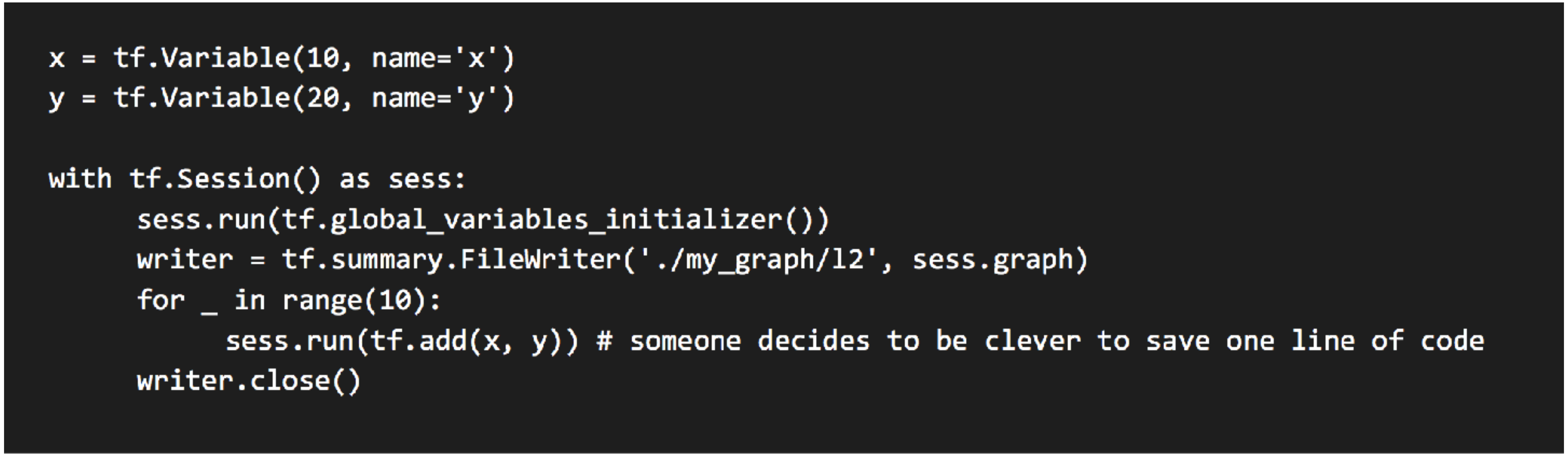
延迟加载

一个正常的例子



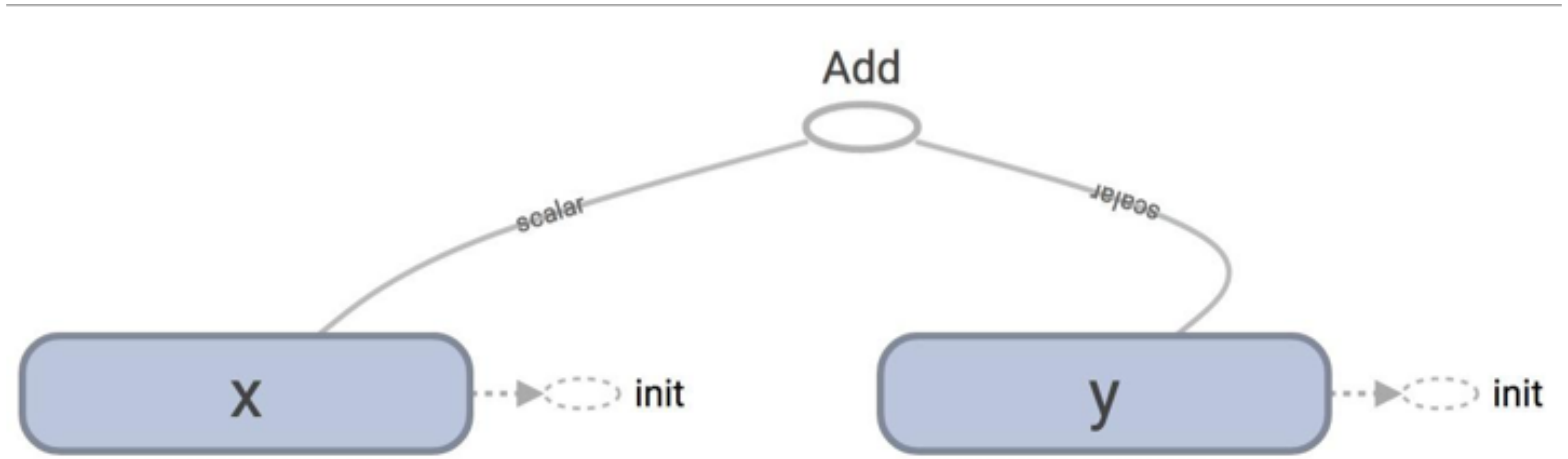
```
x = tf.Variable(10, name='x')
y = tf.Variable(20, name='y')
z = tf.add(x, y) # you create the node for add node before executing the graph

with tf.Session() as sess:
	sess.run(tf.global_variables_initializer())
	writer = tf.summary.FileWriter('./my_graph/l2', sess.graph)
	for _ in range(10):
		sess.run(z)
	writer.close()
```

一个延迟加载的例子

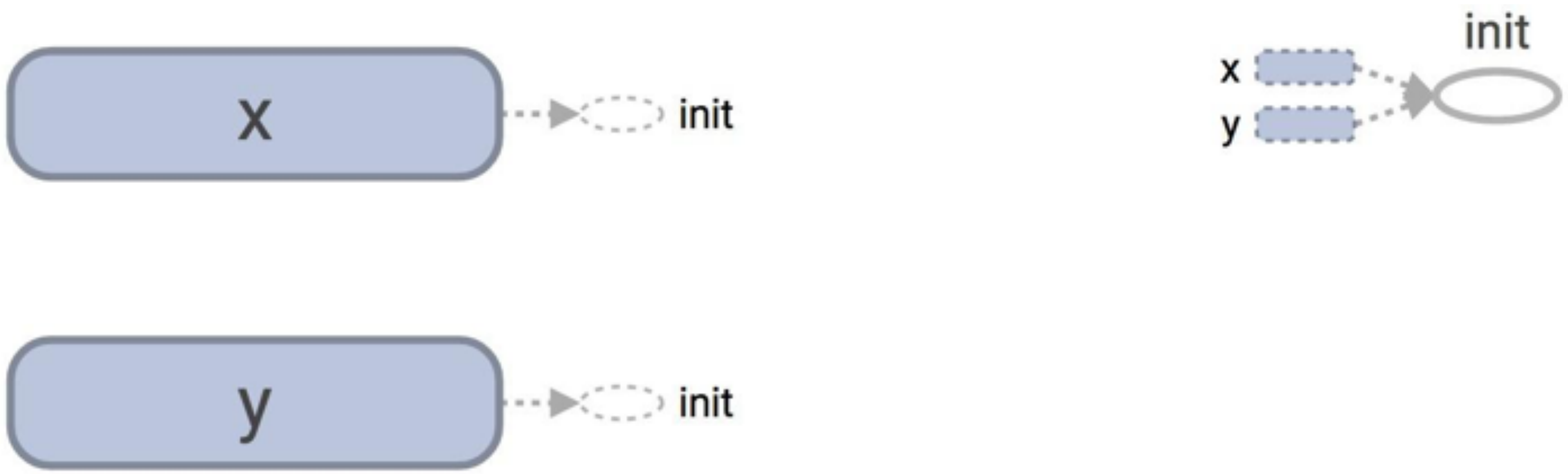


```
x = tf.Variable(10, name='x')
y = tf.Variable(20, name='y')

with tf.Session() as sess:
	sess.run(tf.global_variables_initializer())
	writer = tf.summary.FileWriter('./my_graph/l2', sess.graph)
	for _ in range(10):
		sess.run(tf.add(x, y)) # someone decides to be clever to save one line of code
	writer.close()
```

延迟加载的问题

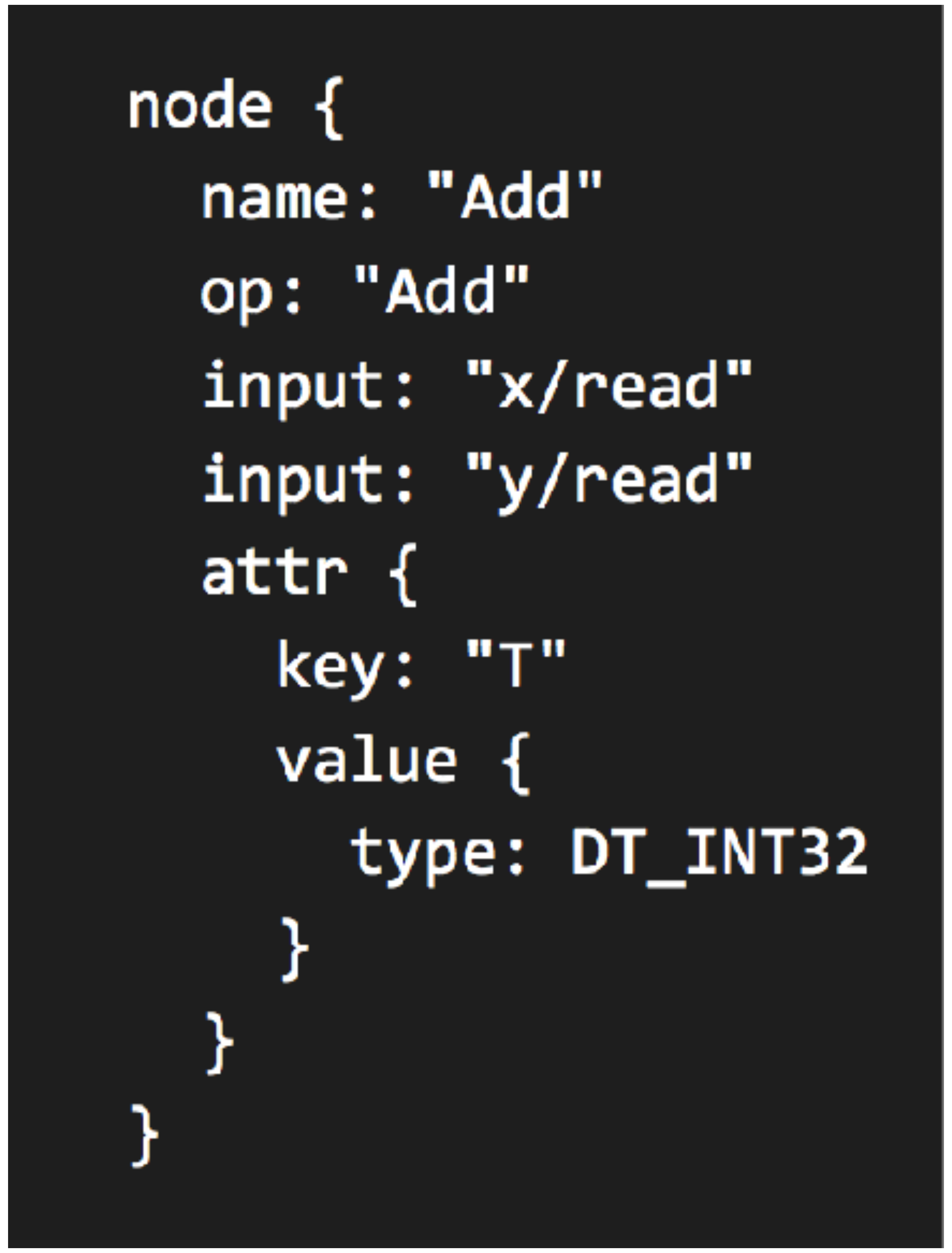


正常加载



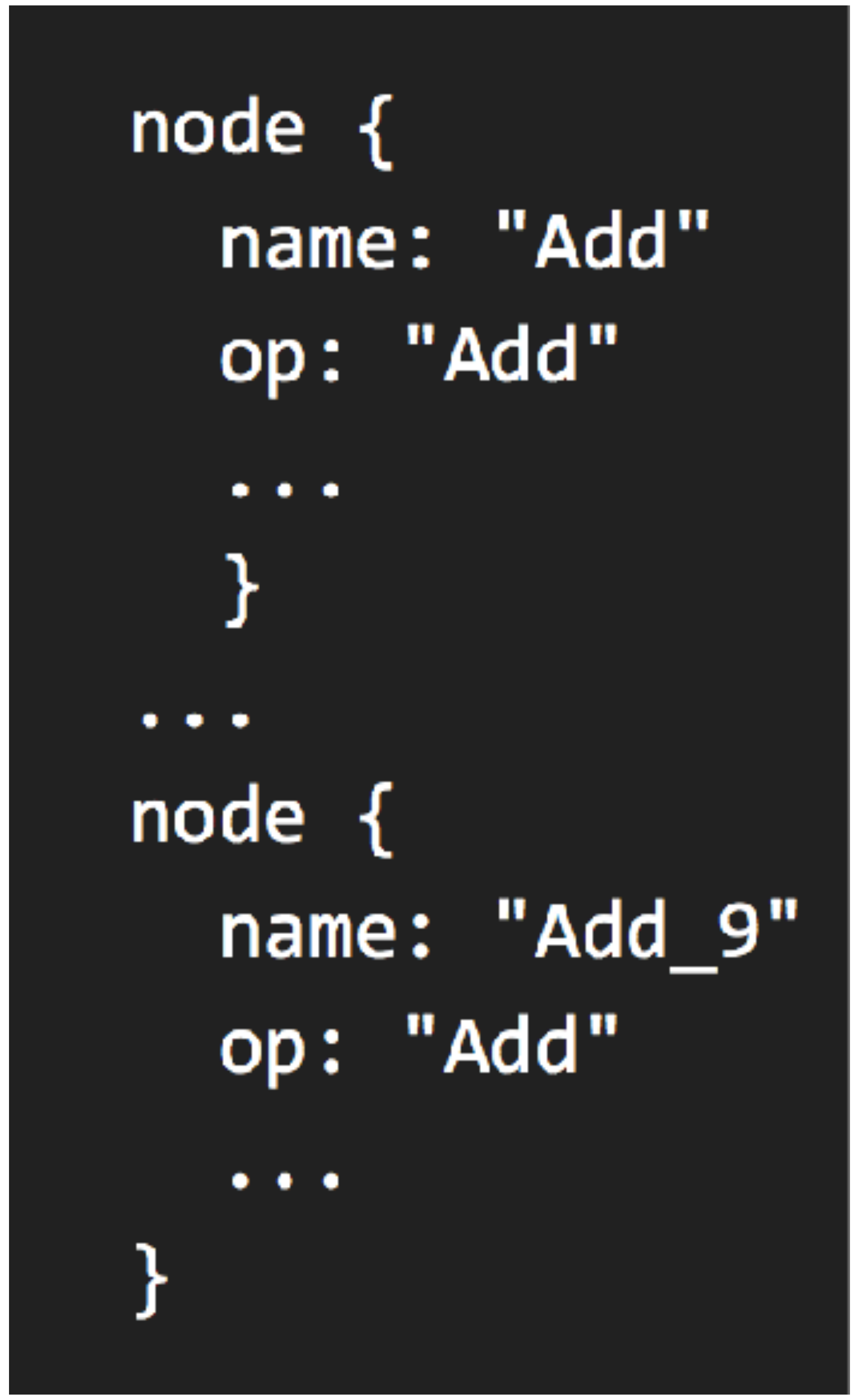
延迟加载

延迟加载的问题



```
node {
  name: "Add"
  op: "Add"
  input: "x/read"
  input: "y/read"
  attr {
    key: "T"
    value {
      type: DT_INT32
    }
  }
}
```

→ 正常加载



```
node {
  name: "Add"
  op: "Add"
  ...
  }
...
node {
  name: "Add_9"
  op: "Add"
  ...
}
```

→ 延迟加载

延迟加载的问题

将会占用更多的资源

本次课程结束

谢谢

如果您有任何问题和建议

请联系
fcbai@thoughtworks.com

ThoughtWorks®