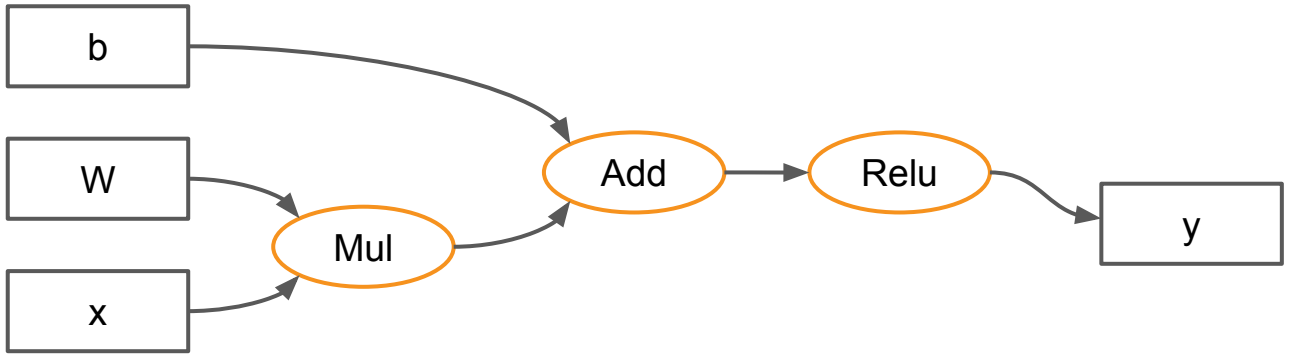
Copyright Notice

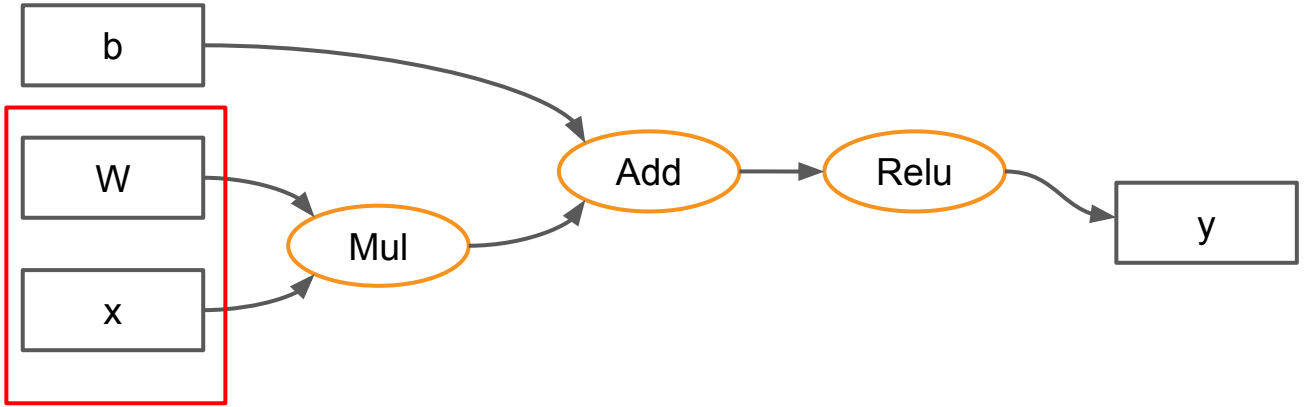
Why graphs?

$$Y = Wx + b$$



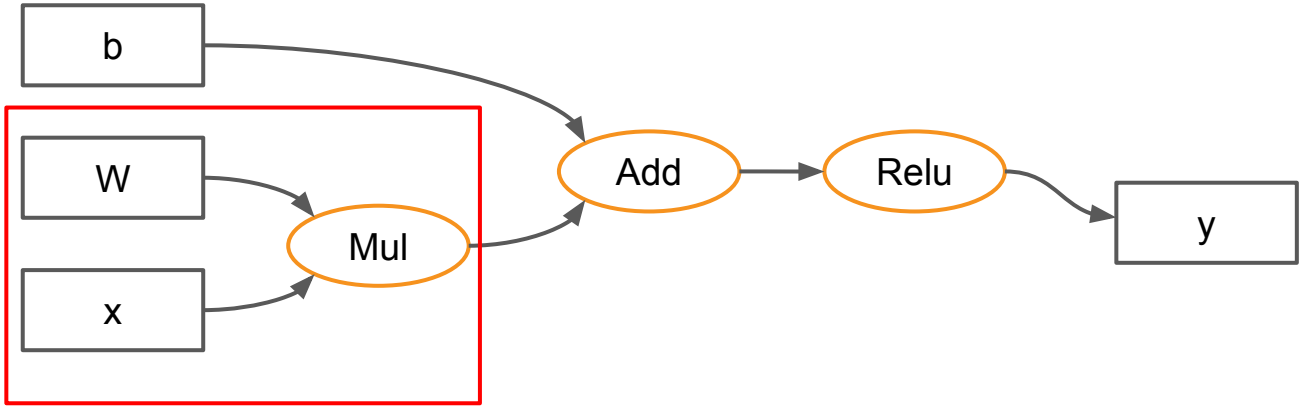
Why graphs?

$$Y = Wx + b$$



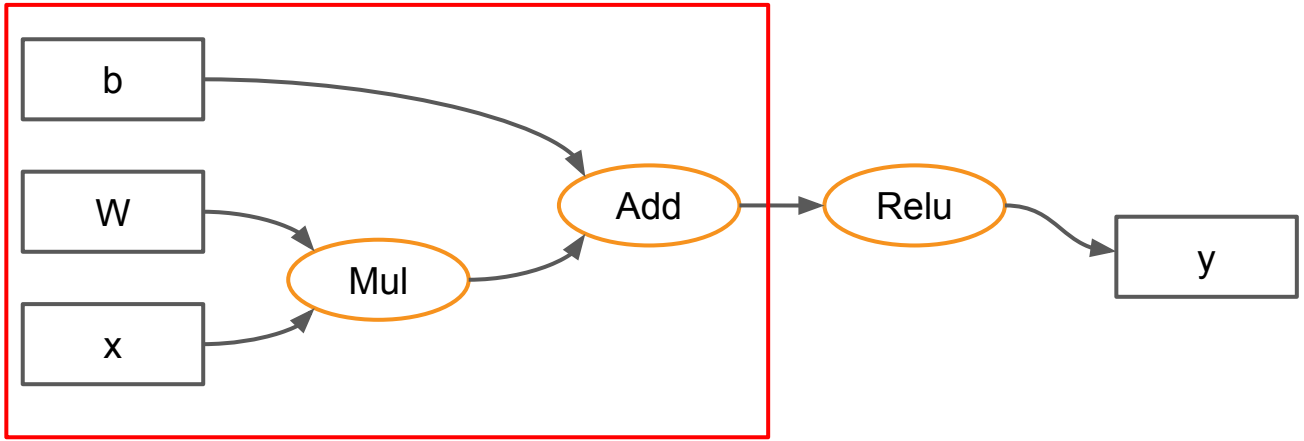
Why graphs?

$$Y = Wx + b$$



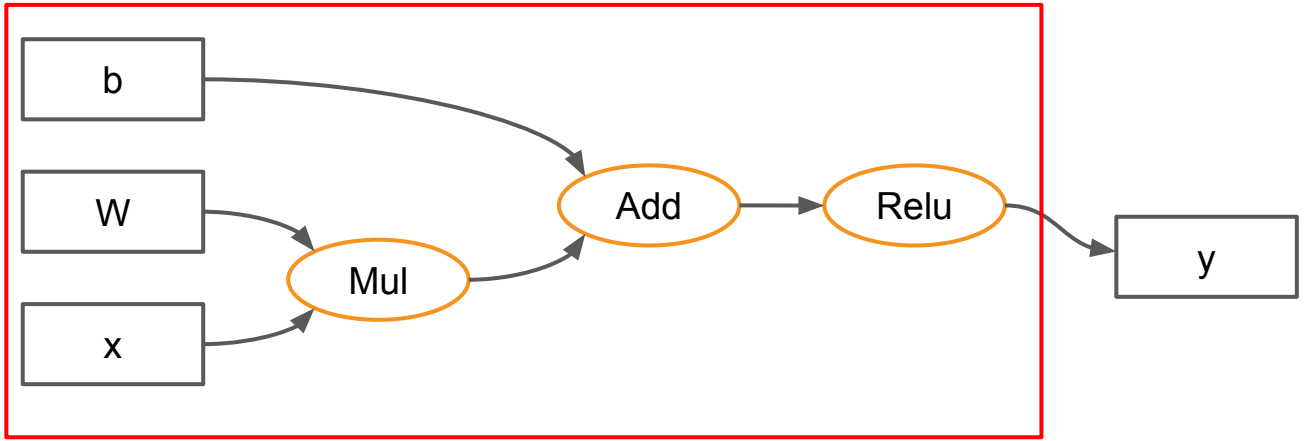
Why graphs?

$$Y = Wx + b$$



Why graphs?

$$Y = Wx + b$$



Eager mode

- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```

Eager mode

- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```

Eager mode

- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```

Eager mode

- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```

Eager mode

- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```

Eager mode

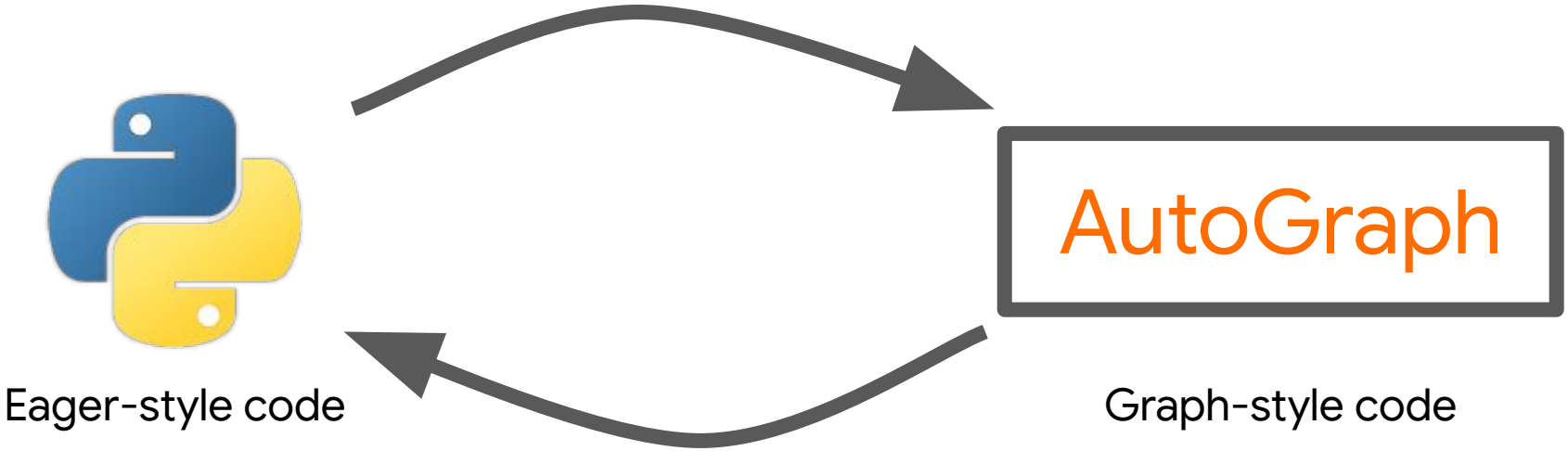
- An intuitive interface
- Easier debugging
- Natural control flow

```
def f(x):
  if x > 0:
    x = x * x
  return x
```

Graph mode

- Parallelism
- Distributed execution
- Compilation
- Portability

```
@tf.function
def f(x):
    def if_true():
        return x * x
    def if_false():
        return x
    x = tf.cond(
          tf.greater(x, 0),
          if_true,
          if_false)
    return x
```


AutoGraph
Eager-style code
Graph-style code

A function as an Op

```
def add(a, b):
  return a + b
```

A function as an Op

```
@tf.function
def add(a, b):
  return a + b
```

A function as an Op

```
print(tf.autograph.to_code(add.python_function))
```

```
def tf__add(a, b):
    do_return = False
    retval_ = ag__.UndefinedReturnValue()
    with ag__.FunctionScope('add', 'fscope',
         ag__.ConversionOptions(recursive=True, user_requested=True,
         optional_features=(), internal_convert_user_code=True)) as fscope:
        try:
            do_return = True
            retval_ = fscope.mark_return_value((a + b))
        except:
            do_return = False
            raise
    (do_return,)
    return ag__.retval(retval_)
```

```
def tf__add(a, b):
    do_return = False
    retval_ = ag__.UndefinedReturnValue()
    with ag__.FunctionScope('add', 'fscope',
         ag__.ConversionOptions(recursive=True, user_requested=True,
         optional_features=(), internal_convert_user_code=True)) as fscope:
        try:
            do_return = True
            retval_ = fscope.mark_return_value((a + b))
        except:
            do_return = False
            raise
    (do_return,)
    return ag__.retval(retval_)
```

Functions have gradients

```
@tf.function
def add(a, b):
  return a + b
```

```
v = tf.Variable(1.0)

with tf.GradientTape() as tape:
  result = add(v, 1.0)

>>> tape.gradient(result, v).numpy()
1.0
```

Chain multiple functions

```
def linear_layer(x):
  return 2*x + 1

@tf.function
def deep_net(x):
  return tf.nn.relu(linear_layer(x))

>>> deep_net(tf.constant((1, 2, 3)))
[3, 5, 7]
```

Chain multiple functions

```
def linear_layer(x):
  return 2*x + 1

@tf.function
def deep_net(x):
  return tf.nn.relu(linear_layer(x))

>>> deep_net(tf.constant((1, 2, 3)))
[3, 5, 7]
```

Functions are polymorphic

```
@tf.function
def double(a):
  return a + a
```

```
>>> double(tf.constant(1)).numpy()
2

>>> double(tf.constant(1.1)).numpy()
2.2

>>> double(tf.constant("a")).numpy()
b'aa'
```

tf.function with Keras

```
class CustomModel(tf.keras.models.Model):
  @tf.function
  def call(self, input_data):
    if tf.reduce_mean(input_data) > 0:
      return input_data
    else:
      return input_data // 2
```

FizzBuzz

"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."

- *http://wiki.c2.com/?FizzBuzzTest*

```
def fizzbuzz(max_num):
  counter = 0
  for num in range(max_num):
    if num % 3 == 0 and num % 5 == 0:
      print('FizzBuzz')
    elif num % 3 == 0:
      print('Fizz')
    elif num % 5 == 0:
      print('Buzz')
    else:
      print(num)
    counter += 1
  return counter
```

FizzBuzz

"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."

- *http://wiki.c2.com/?FizzBuzzTest*

```
def fizzbuzz(max_num):
  counter = 0
  for num in range(max_num):
    if num % 3 == 0 and num % 5 == 0:
      print('FizzBuzz')
    elif num % 3 == 0:
      print('Fizz')
    elif num % 5 == 0:
      print('Buzz')
    else:
      print(num)
    counter += 1
  return counter
```

FizzBuzz

"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."

- *http://wiki.c2.com/?FizzBuzzTest*

```
def fizzbuzz(max_num):
  counter = 0
  for num in range(max_num):
    if num % 3 == 0 and num % 5 == 0:
      print('FizzBuzz')
    elif num % 3 == 0:
      print('Fizz')
    elif num % 5 == 0:
      print('Buzz')
    else:
      print(num)
    counter += 1
  return counter
```

```python
from __future__ import print_function
import tensorflow as tf
import tensorflow.contrib.autograph as ag

def tf__fizzbuzz(max_num):
  with tf.name_scope('fizzbuzz'):
    counter = 0

    def extra_cond(counter_1):
      with tf.name_scope('extra_cond'):
        return True

    def loop_body(num, counter_1):
      with tf.name_scope('loop_body'):

        def if_true_2():
          with tf.name_scope('if_true_2'):
            with ag__.utils.control_dependency_on_returns(ag__.utils.
                dynamic_print('FizzBuzz')):
              return tf.ones(()),

        def if_false_2():
          with tf.name_scope('if_false_2'):

            def if_true_1():
              with tf.name_scope('if_true_1'):
                with ag__.utils.control_dependency_on_returns(ag__.utils.
                    dynamic_print('Fizz')):
                  return tf.ones(()),

            def if_false_1():
              with tf.name_scope('if_false_1'):

                def if_true():
                  with tf.name_scope('if_true'):
                    with ag__.utils.control_dependency_on_returns(ag__.
                        utils.dynamic_print('Buzz')):
                      return tf.ones(()),

                def if_false():
                  with tf.name_scope('if_false'):
                    with ag__.utils.control_dependency_on_returns(ag__.
                        utils.dynamic_print(num)):
                      num_1 = ag__.utils.alias_tensors(num)
                      return tf.ones(()),
                  with ag__.utils.control_dependency_on_returns(ag__.utils.
                      run_cond(tf.equal(num % 5, 0), if_true, if_false)):
                    num_2, if_true, if_false = ag__.utils.alias_tensors(num,
                        if_true, if_false)
                    return tf.ones(()),
              with ag__.utils.control_dependency_on_returns(ag__.utils.
                  run_cond(tf.equal(num % 3, 0), if_true_1, if_false_1)):
                num_3, if_false_1, if_true_1 = ag__.utils.alias_tensors(num,
                    if_false_1, if_true_1)
                return tf.ones(()),
          with ag__.utils.control_dependency_on_returns(ag__.utils.run_cond(
              tf.logical_and(tf.equal(num % 3, 0), tf.equal(num % 5, 0)),
              if_true_2, if_false_2)):
            num_4, if_false_2, if_true_2 = ag__.utils.alias_tensors(num,
                if_false_2, if_true_2)
            counter_1 += 1
            return counter_1,
    counter = ag__.for_loop(ag__.utils.dynamic_builtin(range, max_num),
        extra_cond, loop_body, (counter,))
    return counter
```

Generate AutoGraph code

```
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x

>>> tf.autograph.to_code(f.python_function)
```

```
def tf__f(x):
  do_return = False
  retval_ = ag__.UndefinedReturnValue()
  with ag__.FunctionScope(...) as f_scope:
    def get_state():
      return ()

    def set_state(_):
      pass

    def loop_body(x):
      ...
      return x,

    def loop_test(x):
      return ...
    x, = ag__.while_stmt(loop_test,
                         loop_body,
                         get_state,
                         set_state, ...)
    do_return = True
    retval_ = f_scope.mark_return_value(x)
  do_return,
  return ag__.retval(retval_)
```

Generate AutoGraph code

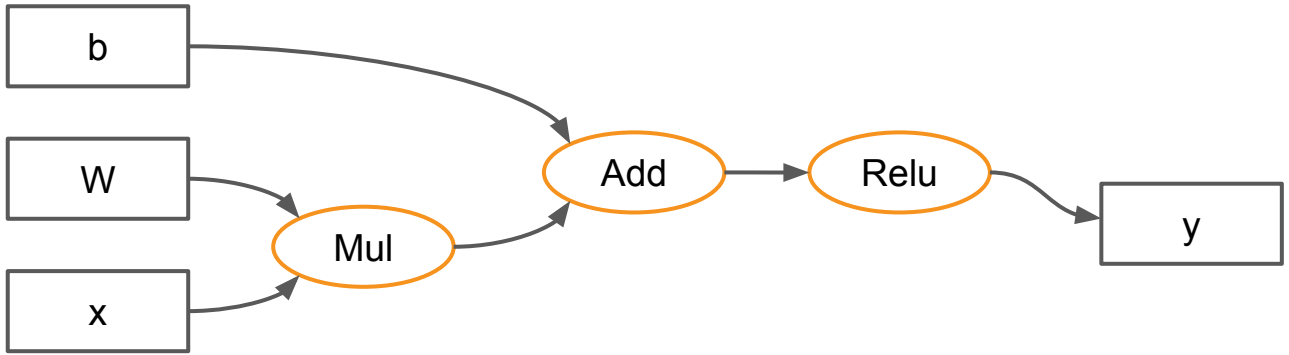
```
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```
>>> tf.autograph.to_code(f.python_function)
```

```
def tf__f(x):
  do_return = False
  retval_ = ag__.UndefinedReturnValue()
  with ag__.FunctionScope(...) as f_scope:
    def get_state():
      return ()

    def set_state(_):
      pass

    def loop_body(x):
      ...
      return x,

    def loop_test(x):
      return ...
    x, = ag__.while_stmt(loop_test,
                         loop_body,
                         get_state,
                         set_state, ...)
    do_return = True
    retval_ = f_scope.mark_return_value(x)
  do_return,
  return ag__.retval(retval_)
```

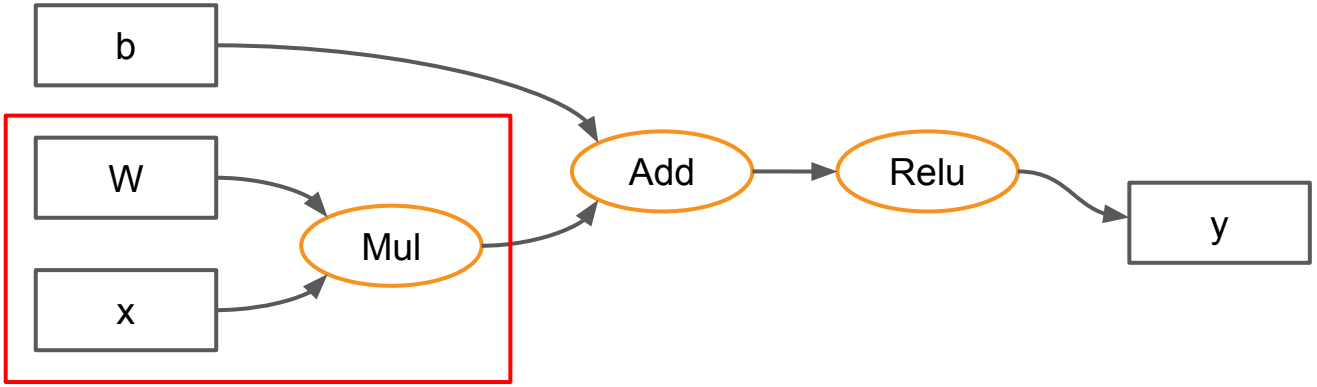
Order of Execution

$$Y = Wx + b$$



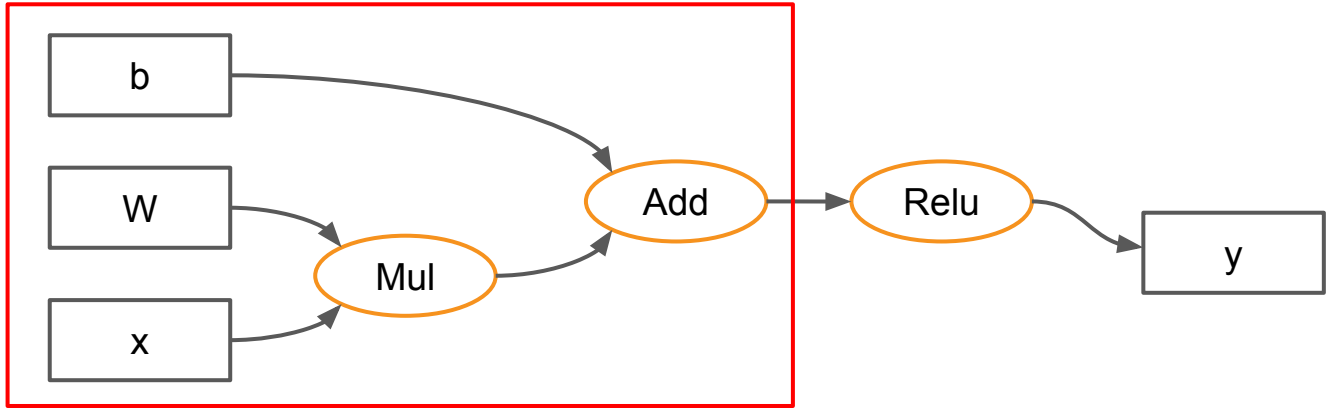
Order of Execution

$$Y = Wx + b$$



Order of Execution

$Y = Wx + b$



Automatic Control Dependencies

```
a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x, y):
  a.assign(y * b)
  b.assign_add(x * a)
  return a + b

>>> f(1.0, 2.0)
10.0
```

Automatic Control Dependencies

```
a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x, y):
  a.assign(y * b)
  b.assign_add(x * a)
  return a + b

>>> f(1.0, 2.0)
10.0
```

Automatic Control Dependencies

```
a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x, y):
  a.assign(y * b)
  b.assign_add(x * a)
  return a + b

>>> f(1.0, 2.0)
10.0
```

a = y * b
a = 2.0 * 2.0
a = 4.0

Automatic Control Dependencies

```
a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x, y):
  a.assign(y * b)
  b.assign_add(x * a)
  return a + b

>>> f(1.0, 2.0)
10.0
```

a = y * b
a = 2.0 * 2.0
a = 4.0

b = 2.0
b = b + x * a
b = 2.0 + 1 * 4.0
b = 6.0

Automatic Control Dependencies

```
a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x, y):
  a.assign(y * b)
  b.assign_add(x * a)
  return a + b

>>> f(1.0, 2.0)
10.0
```

a = y * b
a = 2.0 * 2.0
a = 4.0

b = 2.0
b = b + x * a
b = 2.0 + 1 * 4.0
b = 6.0

a + b
4.0 + 6.0
10.0

Control flows (conditionals)

```
@tf.function
def sign(x):
  if x > 0:
    return 'Positive'
  else:
    return 'Negative'
```

```python
@tf.function
def sign(x):
  if x > 0:
    return 'Positive'
  else:
    return 'Negative'
```

```python
def if_true():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Positive')
    except:
        do_return = False
        raise
    return (retval_, do_return)

def if_false():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Negative')
    except:
        do_return = False
        raise
    return (retval_, do_return)

cond = (x > 0)

(retval_, do_return) = ag__.if_stmt(cond, if_true, if_false,
        get_state, set_state, ('retval_', 'do_return'), ())
```

```python
@tf.function
def sign(x):
  if x > 0:
    return 'Positive'
  else:
    return 'Negative'
```

```python
def if_true():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Positive')
    except:
        do_return = False
        raise
    return (retval_, do_return)

def if_false():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Negative')
    except:
        do_return = False
        raise
    return (retval_, do_return)

cond = (x > 0)

(retval_, do_return) = ag__.if_stmt(cond, if_true, if_false,
         get_state, set_state, ('retval_', 'do_return'), ())
```

```python
@tf.function
def sign(x):
  if x > 0:
    return 'Positive'
  else:
    return 'Negative'
```

```python
def if_true():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Positive')
    except:
        do_return = False
        raise
    return (retval_, do_return)

def if_false():
    try:
        do_return = True
        retval_ = fscope.mark_return_value('Negative')
    except:
        do_return = False
        raise
    return (retval_, do_return)

cond = (x > 0)

(retval_, do_return) = ag__.if_stmt(cond, if_true, if_false,
         get_state, set_state, ('retval_', 'do_return'), ())
```

Control flows (loops)

```
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

Control flows (loops)

```
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```python
def get_state():
    return (x,)

def set_state(loop_vars):
    nonlocal x
    (x,) = loop_vars

def loop_body():
    nonlocal x
    ag__.converted_call(tf.print, (x,), None, fscope)
    x = ag__.converted_call(tf.tanh, (x,), None,
        fscope)

def loop_test():
    return (ag__.converted_call(tf.reduce_sum, (x,),
        None, fscope) > 1)


    ag__.while_stmt(loop_test, loop_body, get_state,
        set_state, ('x',), {})

    try:
        do_return = True
        retval_ = fscope.mark_return_value(x)
    Except:
        do_return = False
        raise
```

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```python
def get_state():
    return (x,)

def set_state(loop_vars):
    nonlocal x
    (x,) = loop_vars

def loop_body():
    nonlocal x
    ag__.converted_call(tf.print, (x,), None, fscope)
    x = ag__.converted_call(tf.tanh, (x,), None,
        fscope)

def loop_test():
    return (ag__.converted_call(tf.reduce_sum, (x,),
        None, fscope) > 1)


    ag__.while_stmt(loop_test, loop_body, get_state,
        set_state, ('x',), {})

    try:
        do_return = True
        retval_ = fscope.mark_return_value(x)
    Except:
        do_return = False
        raise
```

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```python
def get_state():
    return (x,)

def set_state(loop_vars):
    nonlocal x
    (x,) = loop_vars

def loop_body():
    nonlocal x
    ag__.converted_call(tf.print, (x,), None, fscope)
    x = ag__.converted_call(tf.tanh, (x,), None,
        fscope)

def loop_test():
    return (ag__.converted_call(tf.reduce_sum, (x,),
        None, fscope) > 1)


    ag__.while_stmt(loop_test, loop_body, get_state,
        set_state, ('x',), {})

    try:
        do_return = True
        retval_ = fscope.mark_return_value(x)
    Except:
        do_return = False
        raise
```

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x
```

```python
def get_state():
    return (x,)

def set_state(loop_vars):
    nonlocal x
    (x,) = loop_vars

def loop_body():
    nonlocal x
    ag__.converted_call(tf.print, (x,), None, fscope)
    x = ag__.converted_call(tf.tanh, (x,), None,
        fscope)

def loop_test():
    return (ag__.converted_call(tf.reduce_sum, (x,),
        None, fscope) > 1)


    ag__.while_stmt(loop_test, loop_body, get_state,
        set_state, ('x',), {})

    try:
        do_return = True
        retval_ = fscope.mark_return_value(x)
    Except:
        do_return = False
        raise
```

Default behavior of tracing variables (eager mode)

```python
def f(x):
  print("Traced with", x)

for i in range(5):
  f(2)
```

```
Traced with 2
Traced with 2
Traced with 2
Traced with 2
Traced with 2
```

Default behavior of tracing variables (eager mode)

```python
def f(x):
  print("Traced with", x)

for i in range(5):
  f(2)
```

```
Traced with 2
Traced with 2
Traced with 2
Traced with 2
Traced with 2
```

Use TensorFlow Ops to trace (graph mode)

```
@tf.function
def f(x):
  print("Traced with", x)
  tf.print("Executed with", x)

for i in range(5):
  f(2)
```

```
Traced with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
```

Use TensorFlow Ops to trace (graph mode)

```
@tf.function
def f(x):
  print("Traced with", x)
  tf.print("Executed with", x)

for i in range(5):
  f(2)
```

```
Traced with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
```

Use TensorFlow Ops to trace (graph mode)

```
@tf.function
def f(x):
  print("Traced with", x)
  tf.print("Executed with", x)

for i in range(5):
  f(2)
```

```
Traced with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
Executed with 2
```

Dangerous variable creation behavior

```
@tf.function
def f(x):
  v = tf.Variable(1.0)
  v.assign_add(x)
  return v
```

```
>>> f(1)

Caught expected exception
  <class 'ValueError'>: in
converted code:
...
```

Declare outside the function

```
v = tf.Variable(1.0)
@tf.function
def f(x):
  v.assign_add(x)
  return v
```