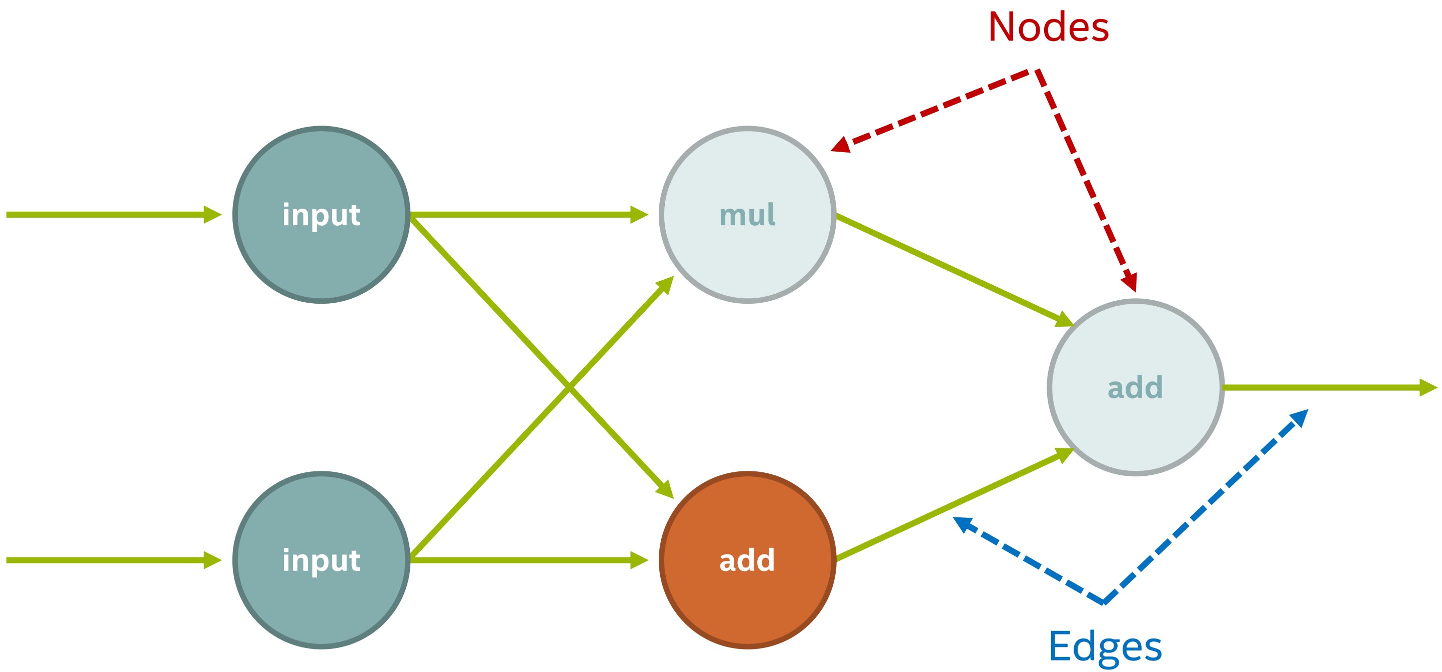
Intel Nervana™ AI Academy

TENSORFLOW FUNDAMENTALS

WHAT IS TENSORFLOW?

- Framework for math using computation graphs
- Has features specifically for machine learning
- Primary interface is Python, integrates with NumPy
  - Implemented in C++ and CUDA
- Designed to be flexible, scalable, and deployable

COMPUTATION GRAPH

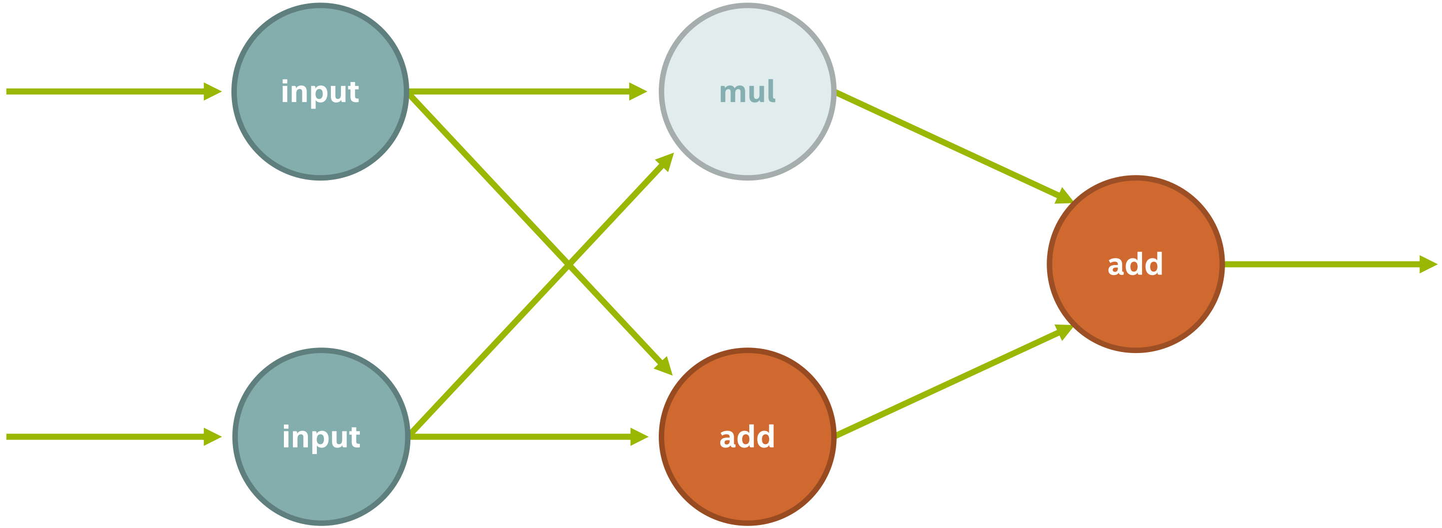


TENSORFLOW HISTORY

- Developed by Google, successor to DistBelief
- Open source implementation released in November 2015
- Key developments
  - **TensorFlow Serving**; Feb 2016
  - **Distributed runtime**; April 2016
  - **Accelerated Linear Algebra (XLA)**; January 2017
  - **TensorFlow Fold**; February 2017
- Current version is 1.0.0
  - Code guaranteed to be backwards-compatible until 2.0.0

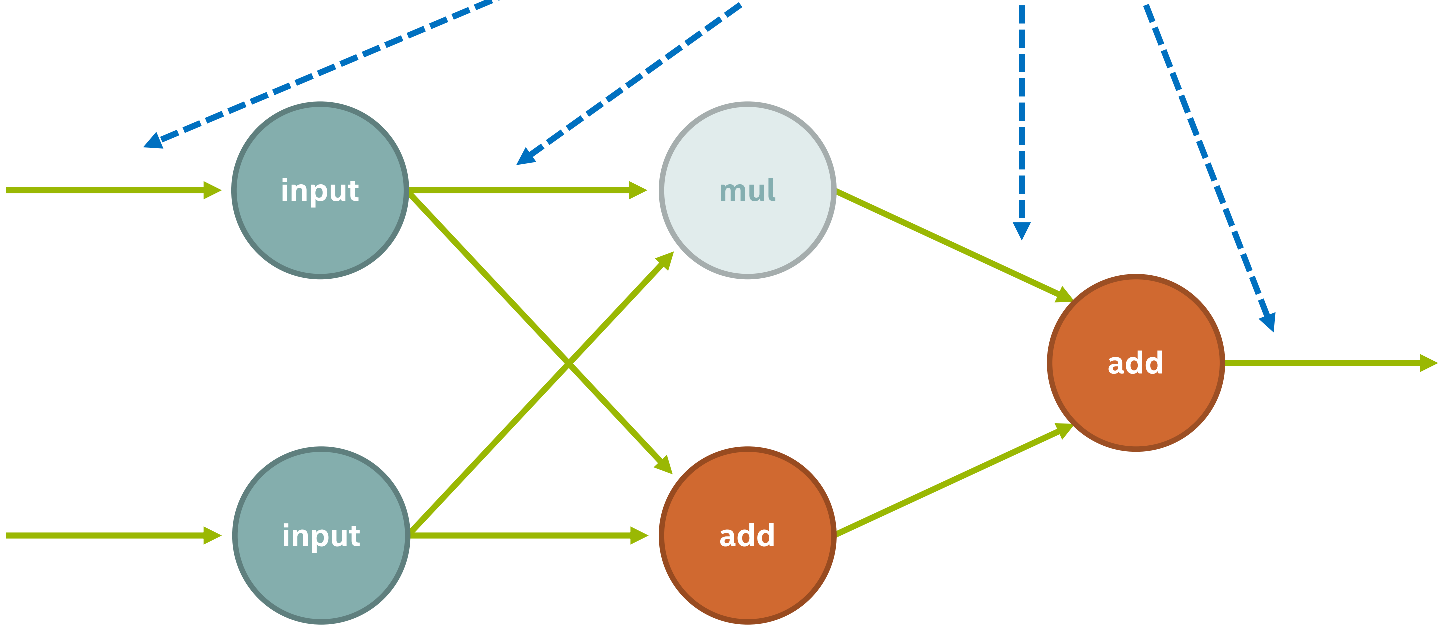
COMPUTATION GRAPHS

DATA FLOW

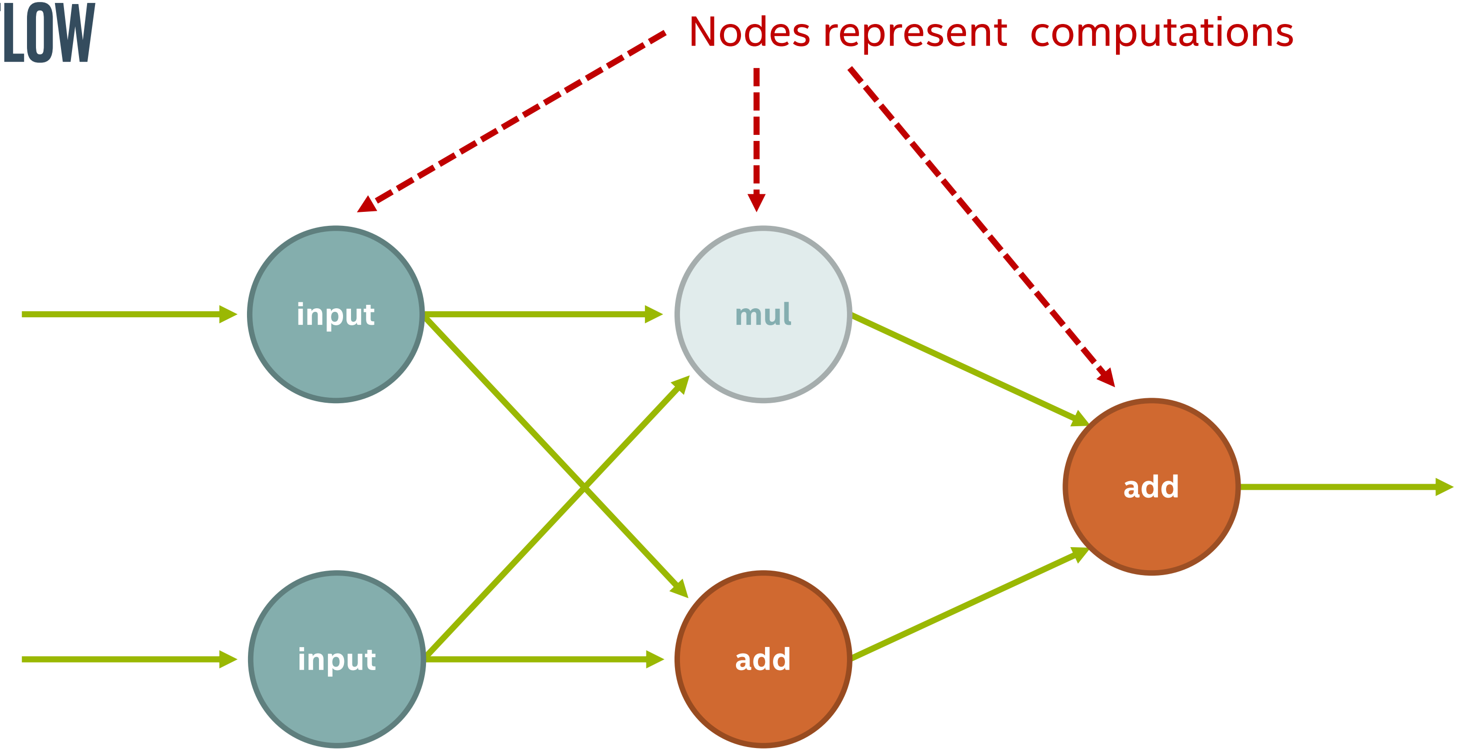


DATA FLOW

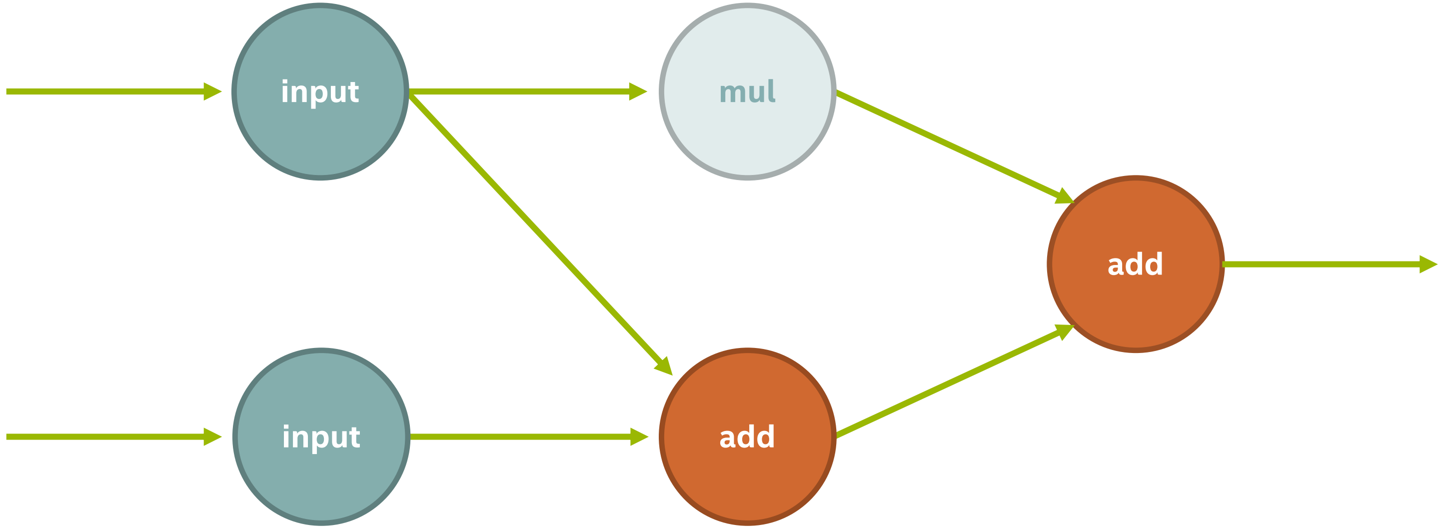
Edges represent numerical data flowing through the graph



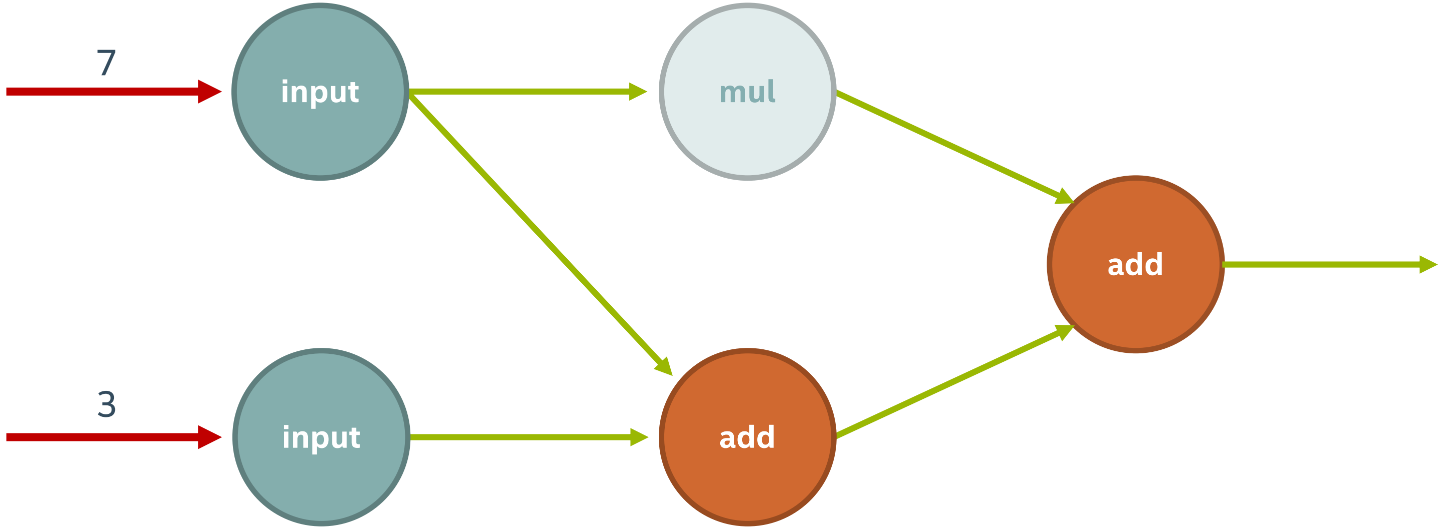
DATA FLOW



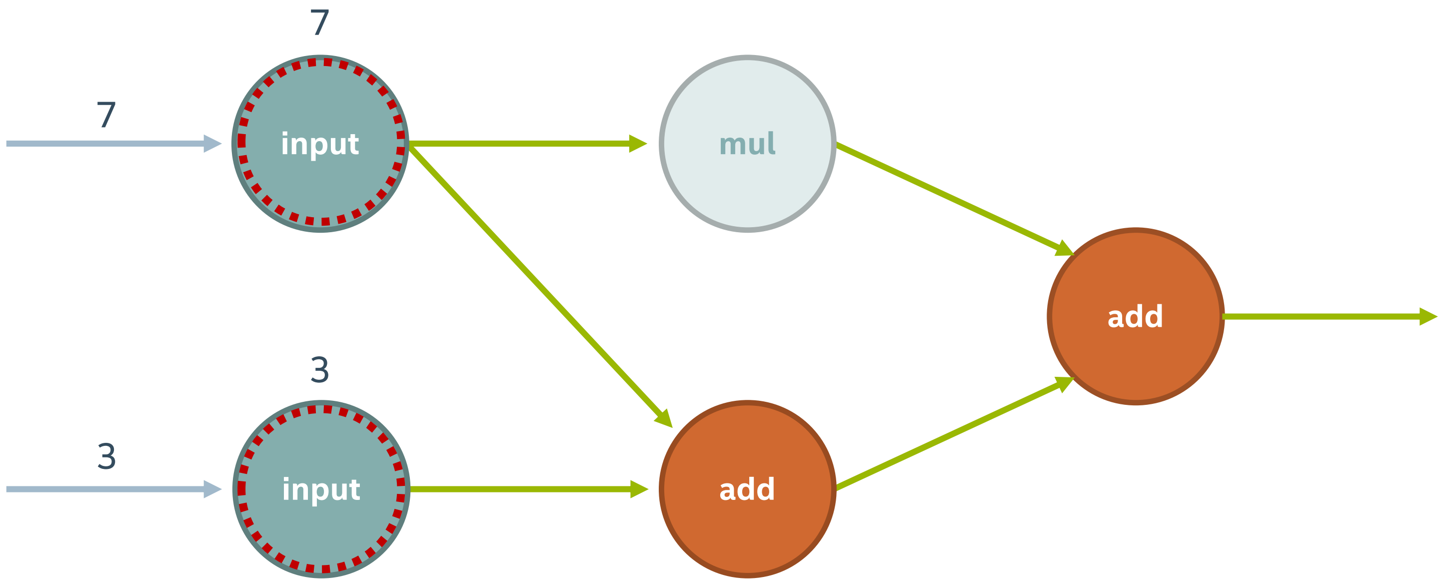
DATA FLOW



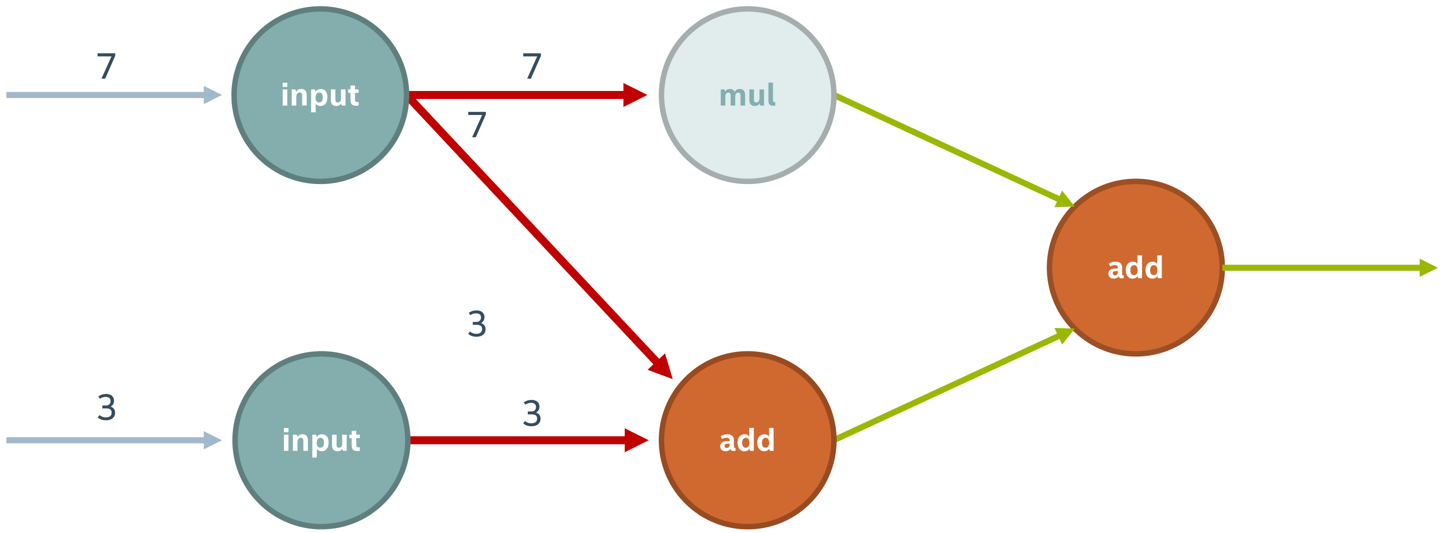
DATA FLOW



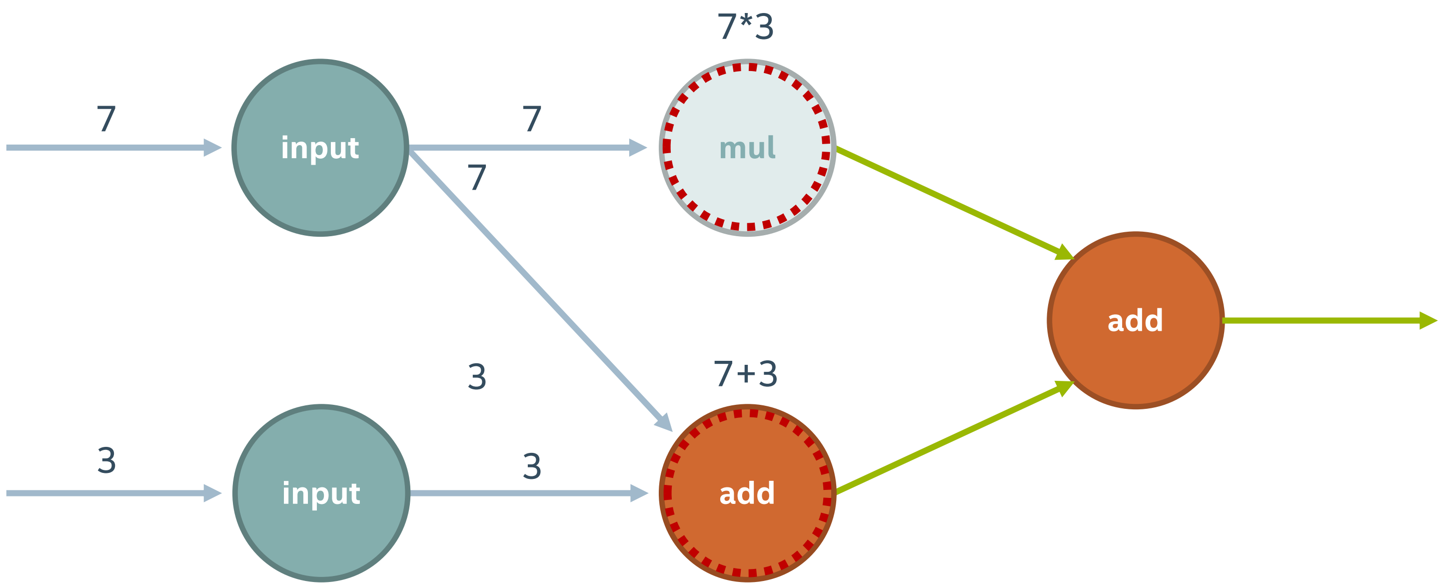
DATA FLOW



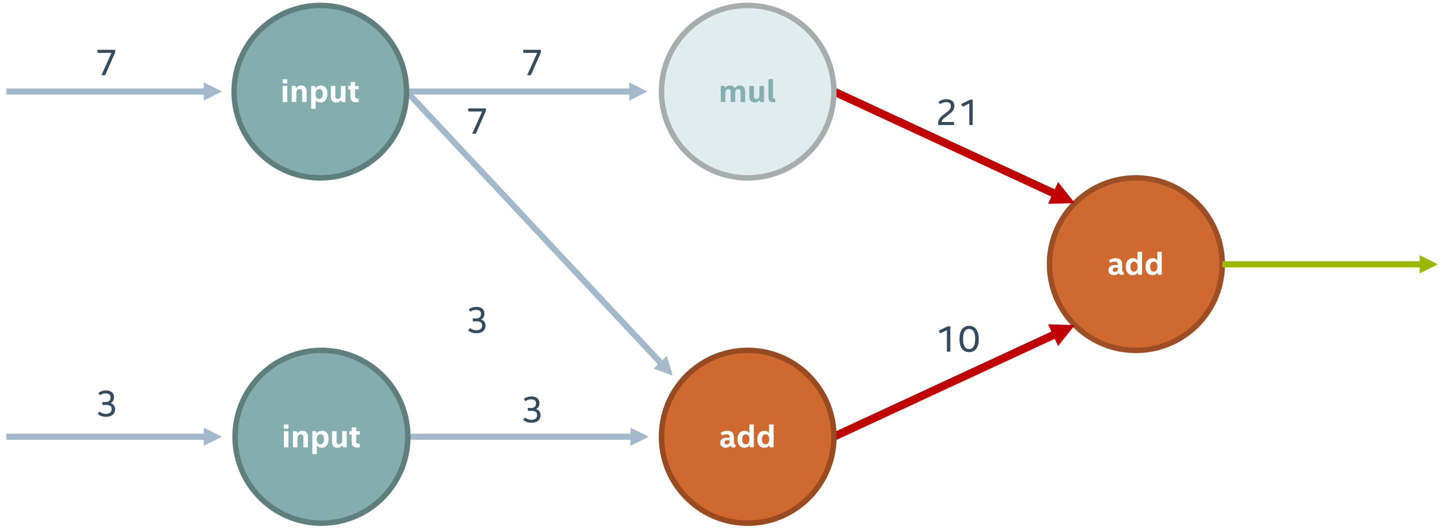
DATA FLOW



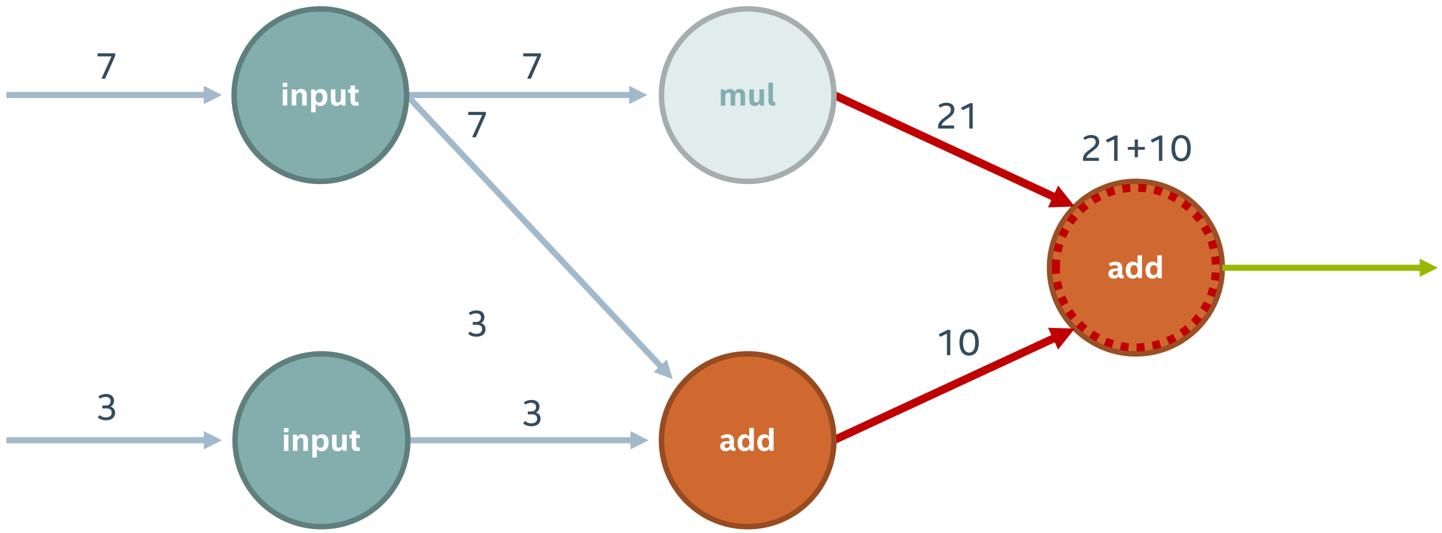
DATA FLOW



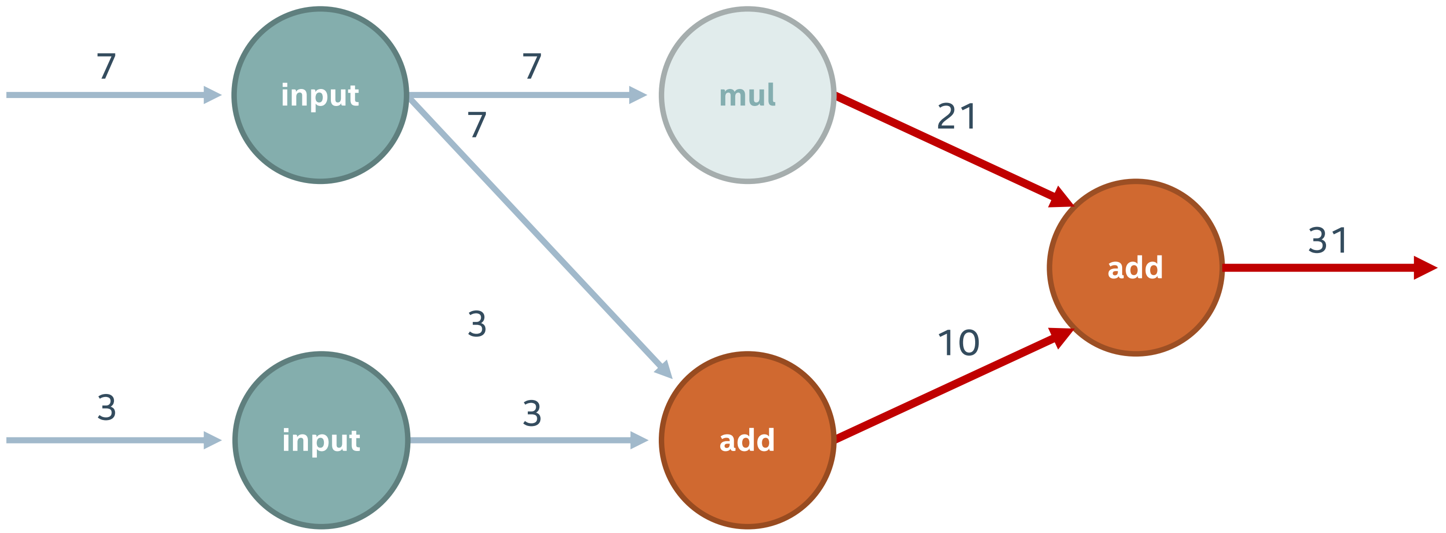
DATA FLOW



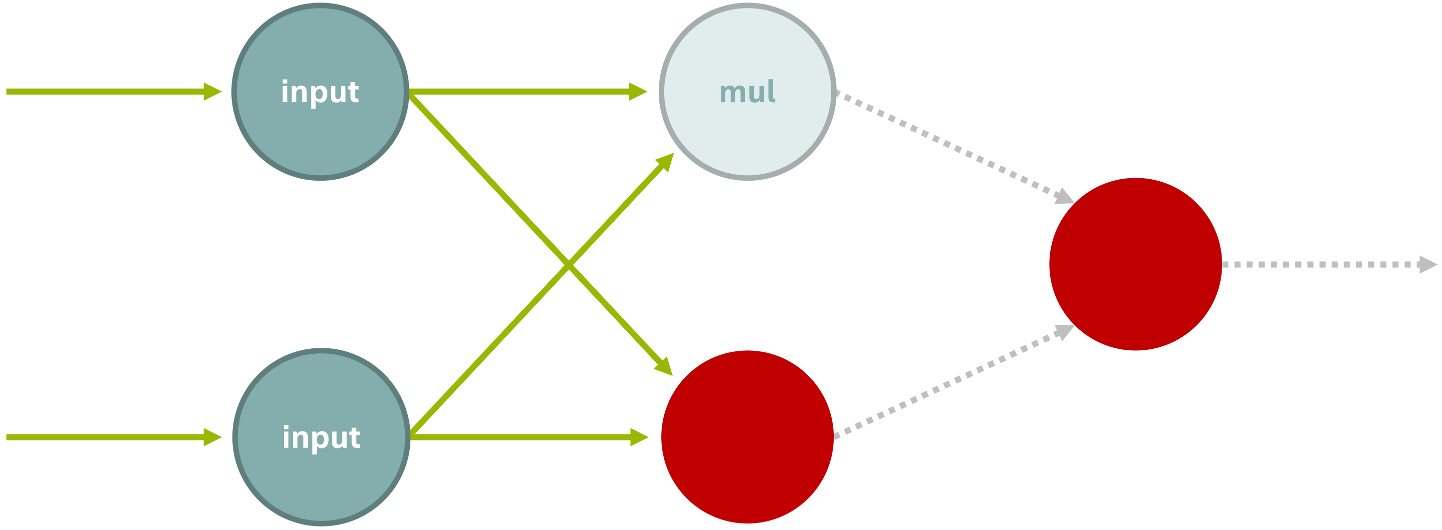
DATA FLOW



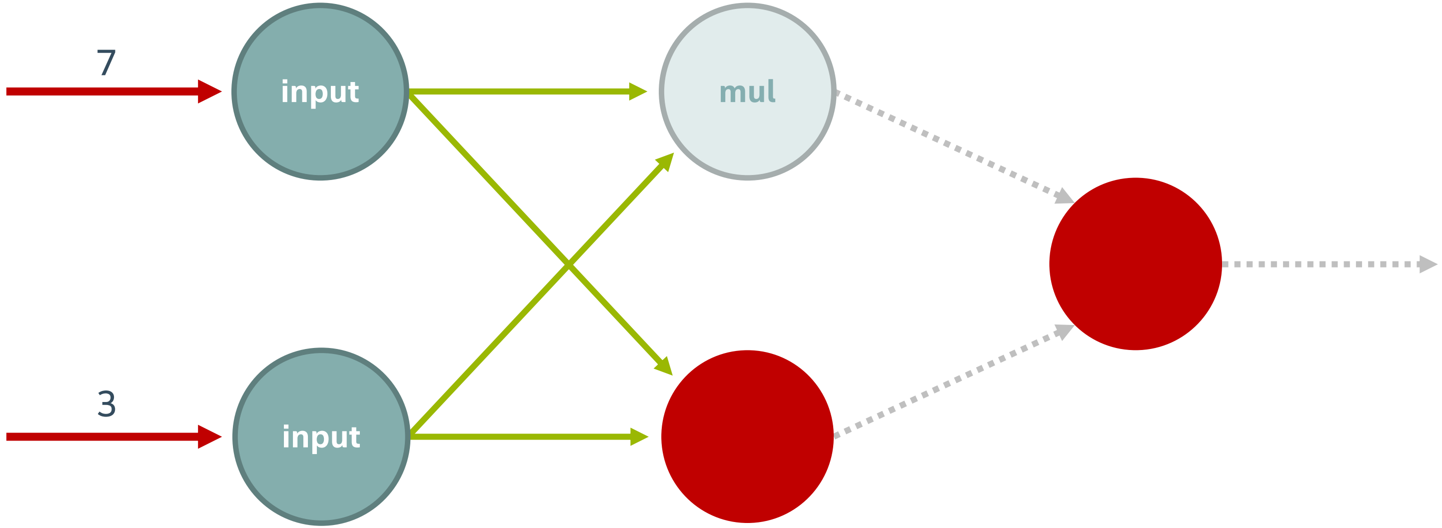
DATA FLOW



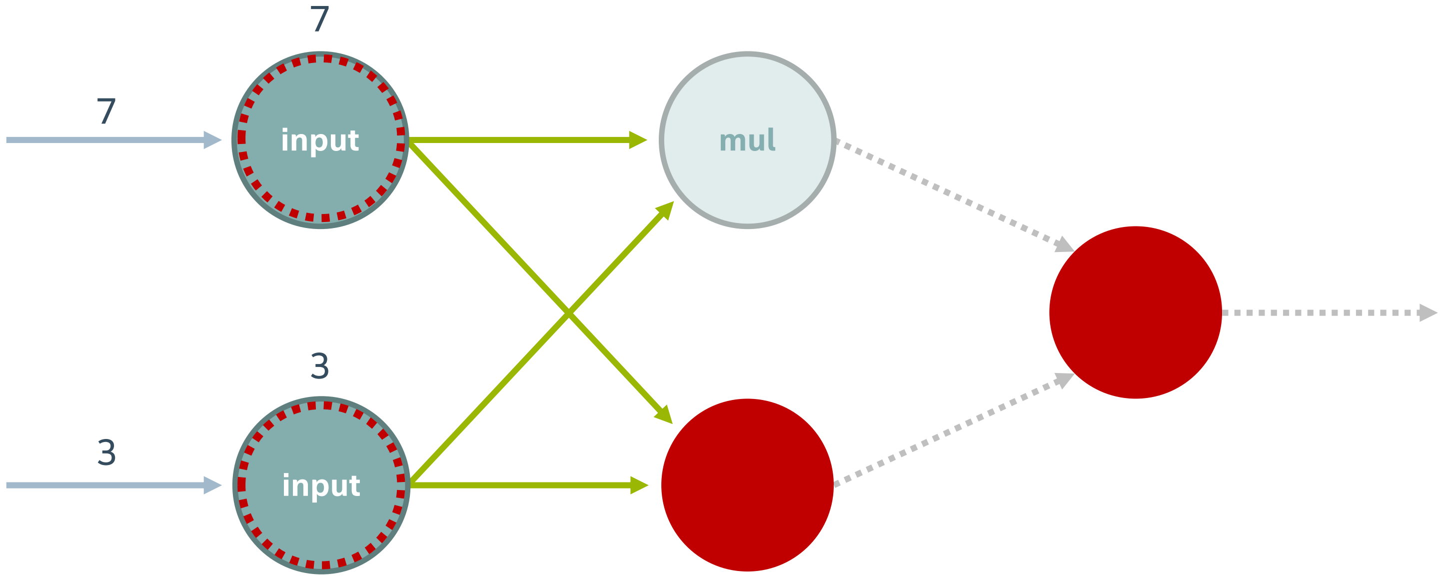
PARTIAL EXECUTION



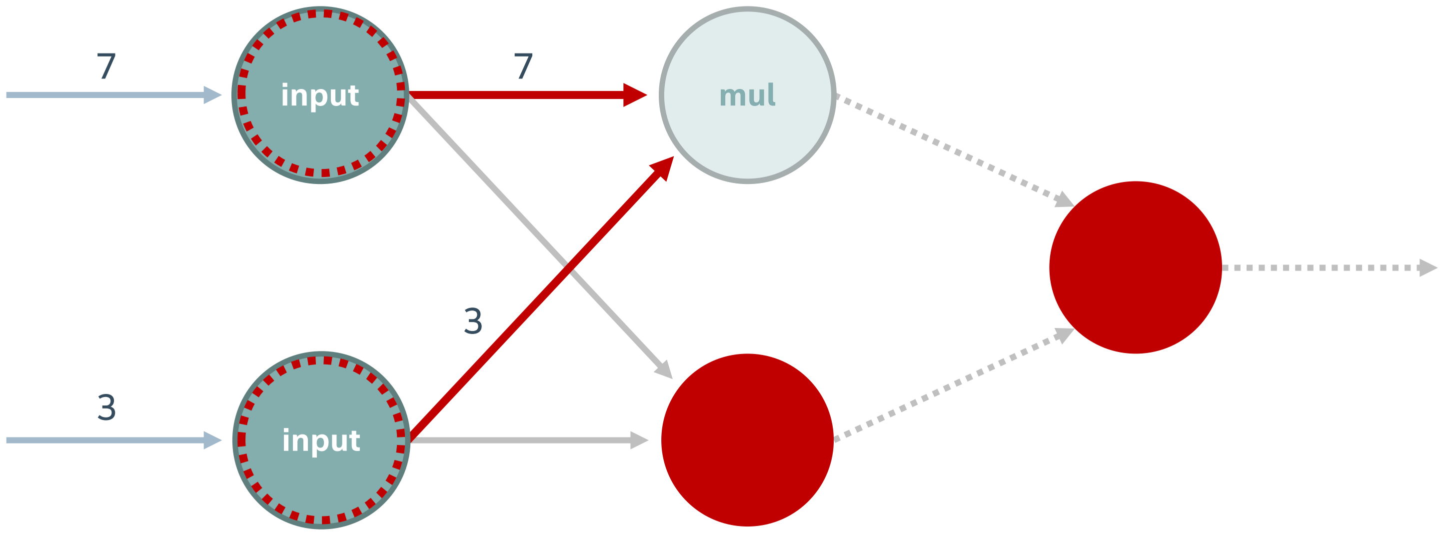
PARTIAL EXECUTION



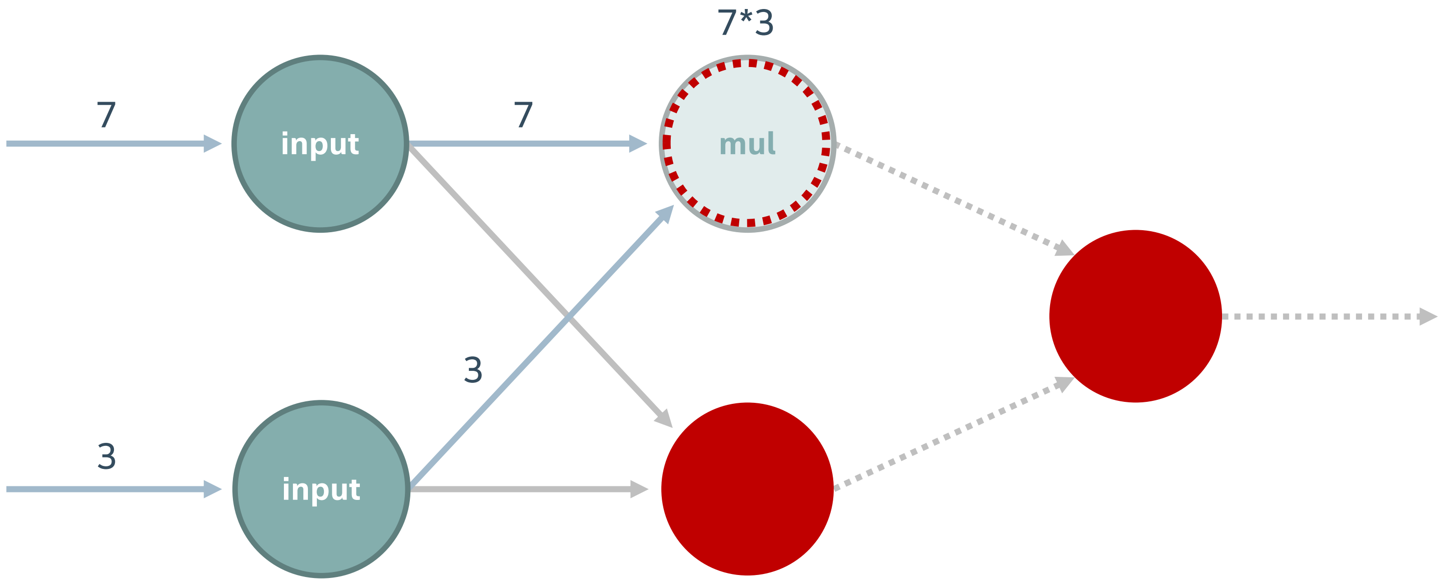
PARTIAL EXECUTION



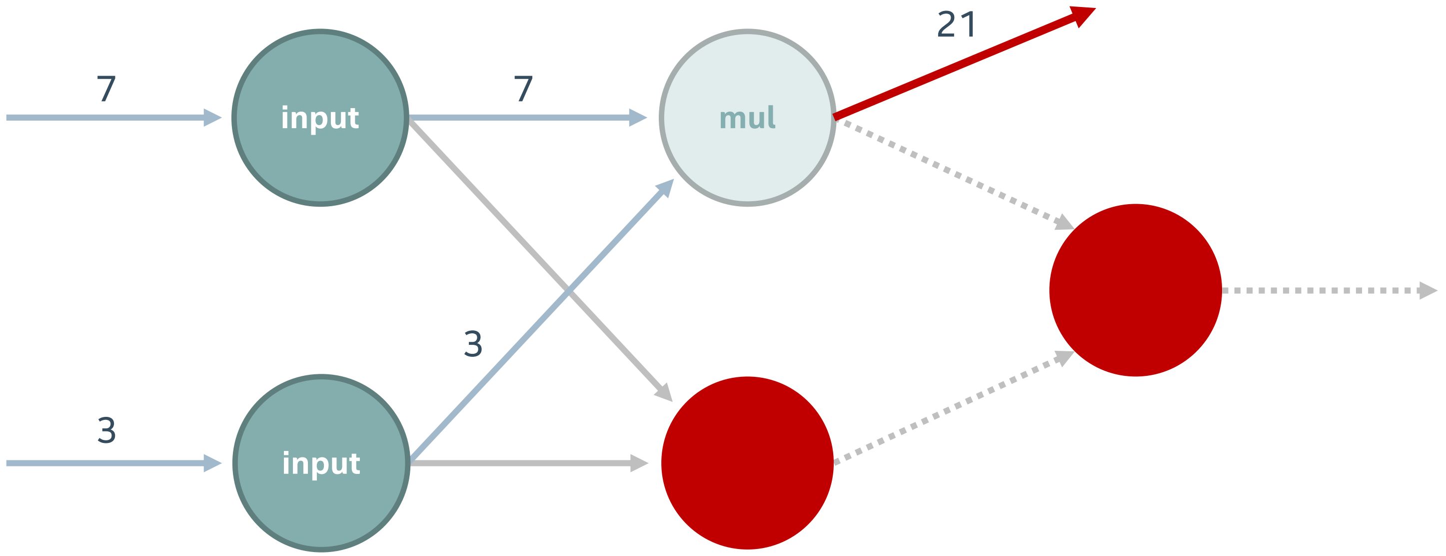
PARTIAL EXECUTION



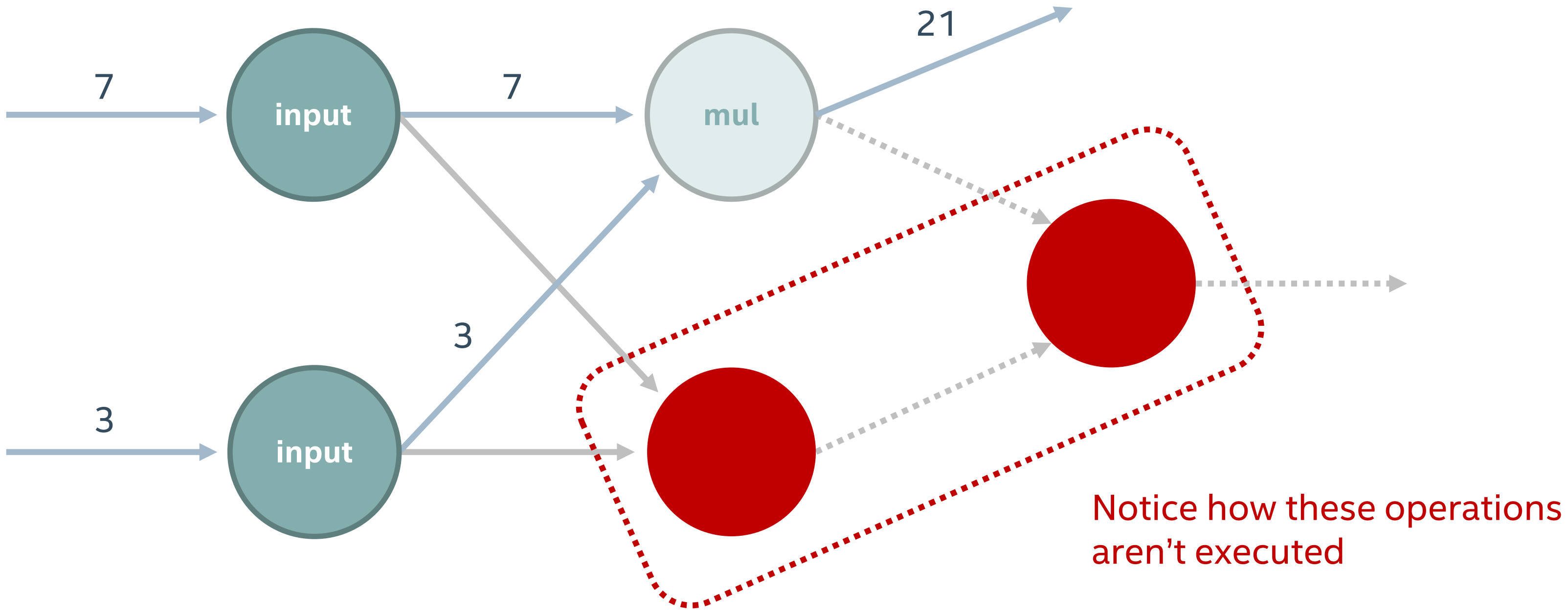
PARTIAL EXECUTION



PARTIAL EXECUTION



PARTIAL EXECUTION

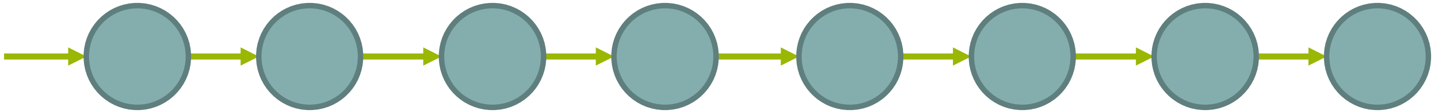


DEPENDENCIES

Nodes which are needed to compute another node

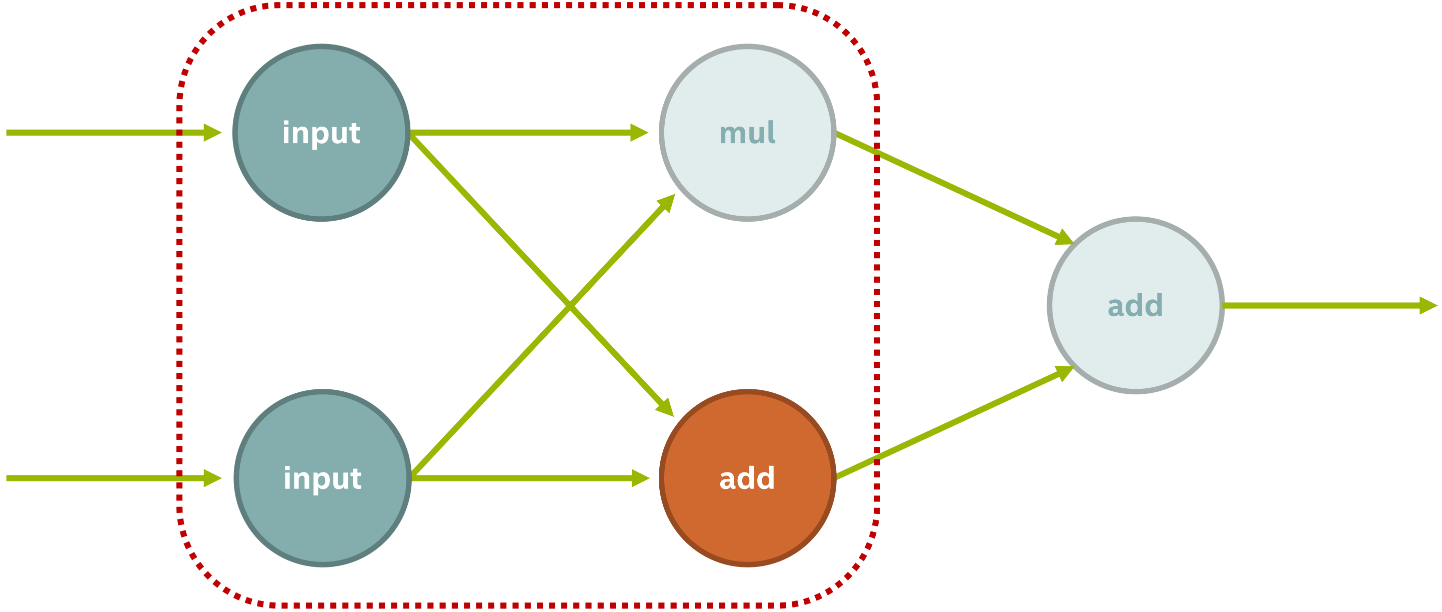
Different nodes have different dependencies

Nodes with *no* dependencies can run at any time



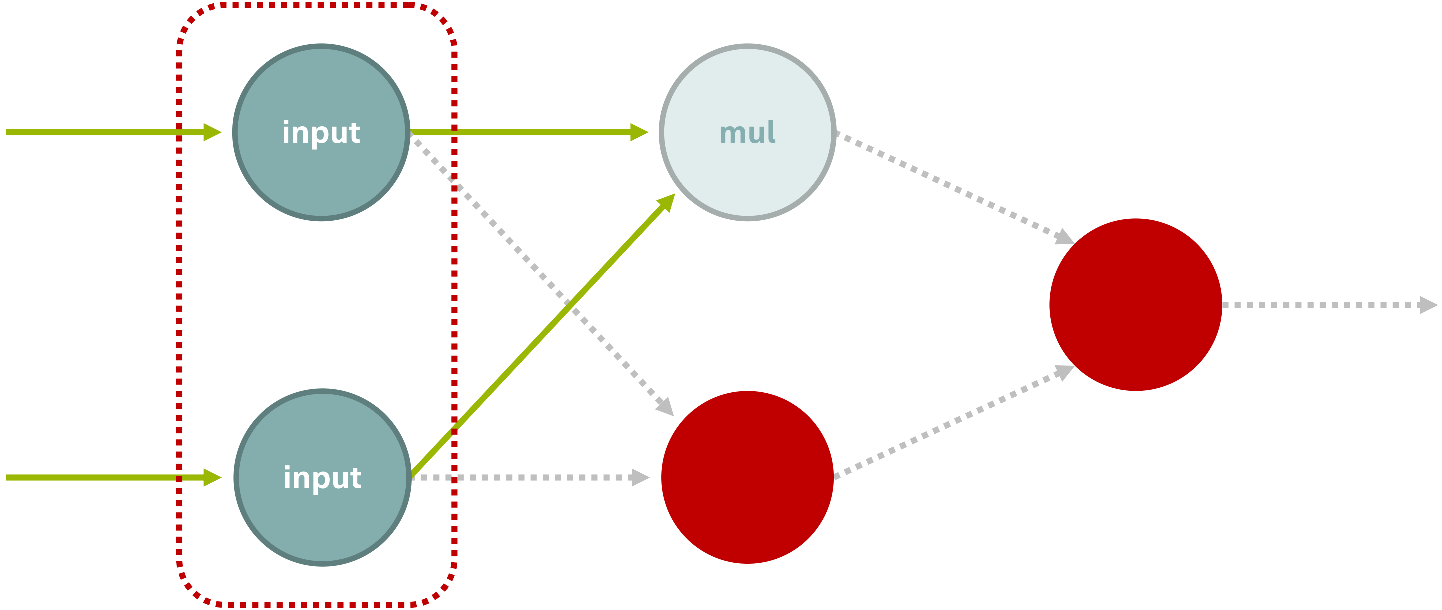
DEPENDENCIES

Dependencies of the final add node



DEPENDENCIES

Dependencies of the mul node



TENSORFLOW FUNDAMENTALS

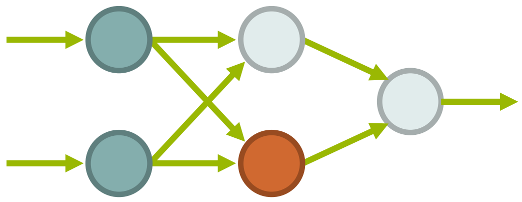
TODAY'S ITINERARY

Primary use-pattern: Define and Run

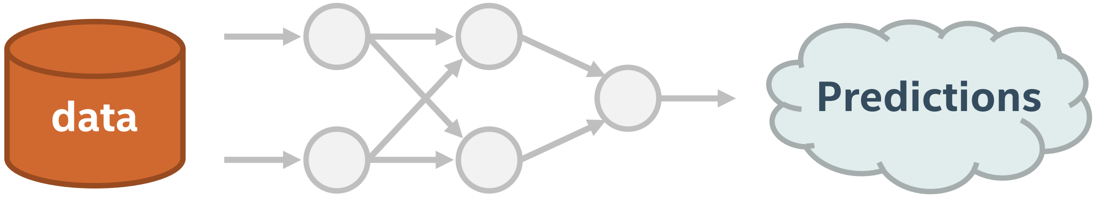
Quick example

TWO-STEP PROGRAMMING PATTERN

1. Define a computation graph



2. Run the graph



DEFINING GRAPHS

MAIN TENSORFLOW API CLASSES

Graph

- Container for operations and tensors

Operation

- Nodes in the graph
- Represent computations

Tensor

- Edges in the graph
- Represent data

When you import TensorFlow, it automatically creates a default graph

```
>>> import tensorflow as tf
```

default

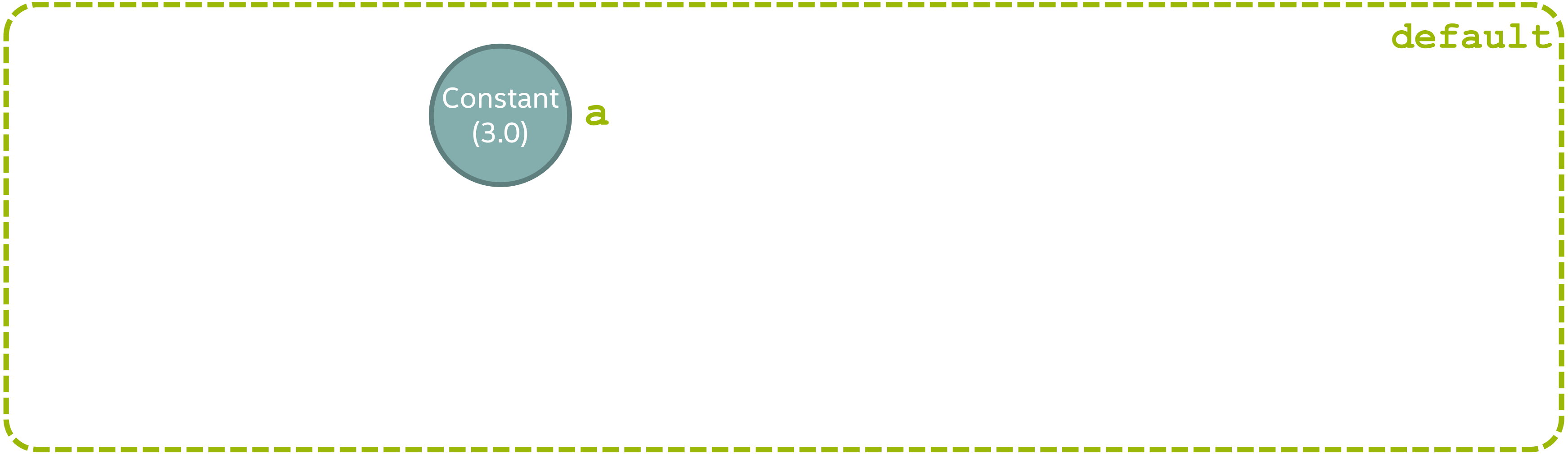
This is the default location for model operations

```
>>> import tensorflow as tf
```

default

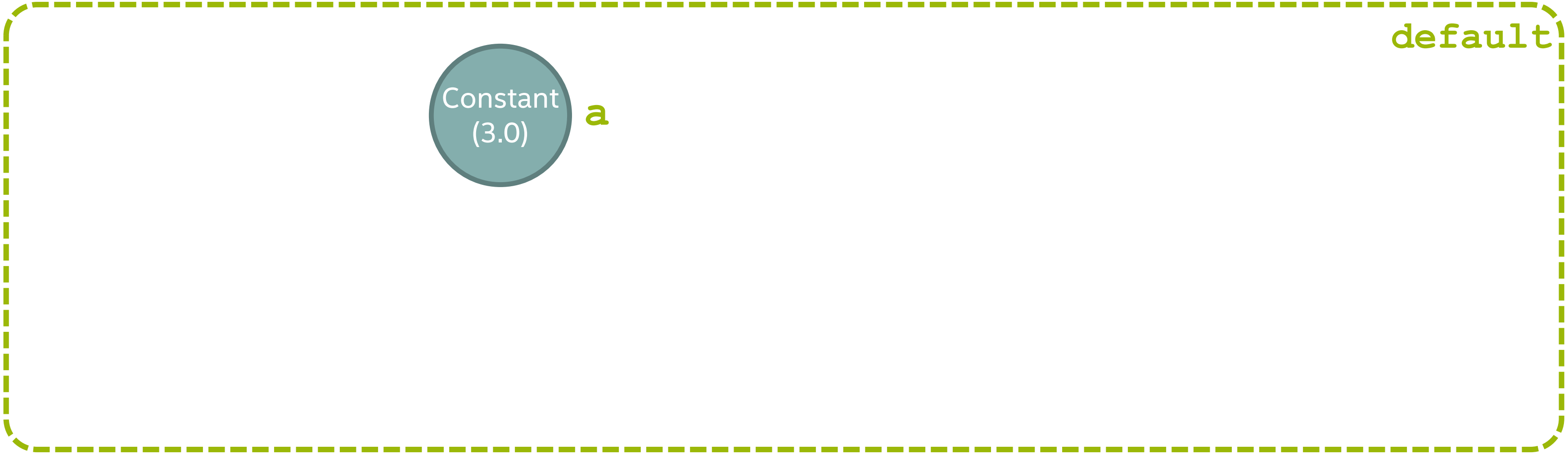
We can create constant data values with `tf.constant()`

```
>>> a = tf.constant(3.0)
```



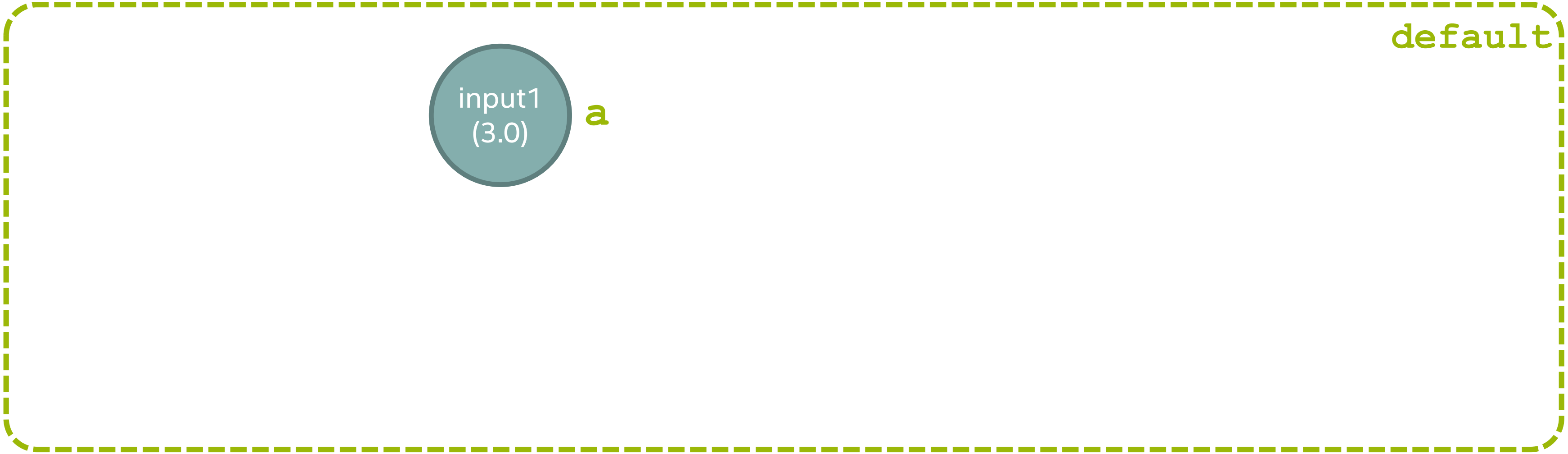
`tf.constant()` creates an `Operation` that returns a fixed value

```
>>> a = tf.constant(3.0)
```



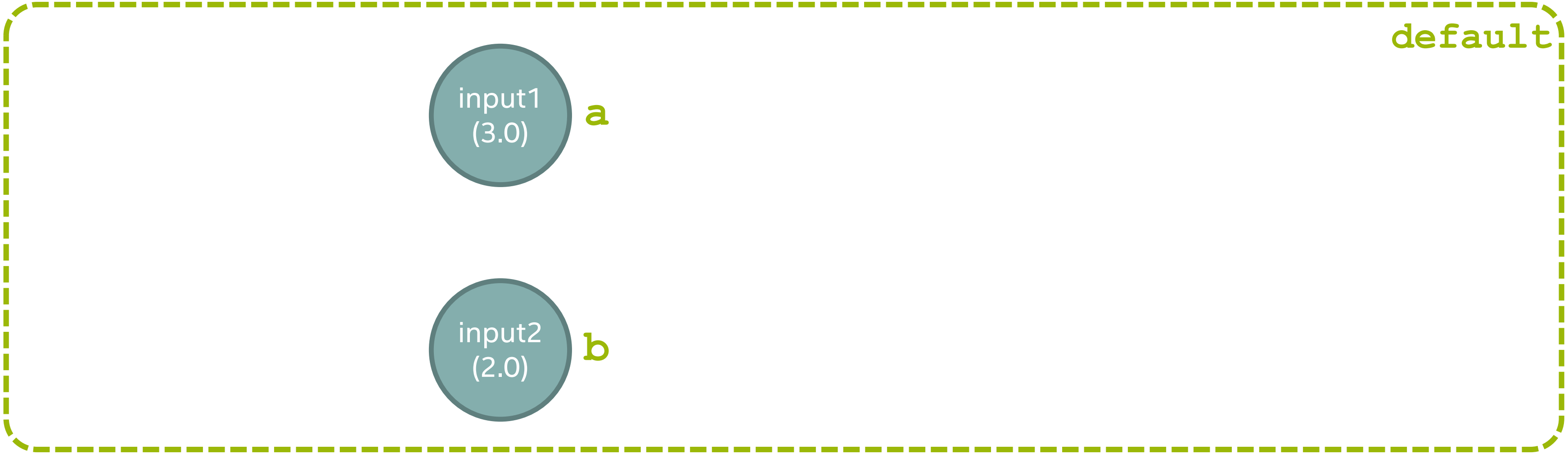
`Operation` functions can be given a `name` parameter, which gives the Operation a string name

```
>>> a = tf.constant(3.0, name="input1")
```



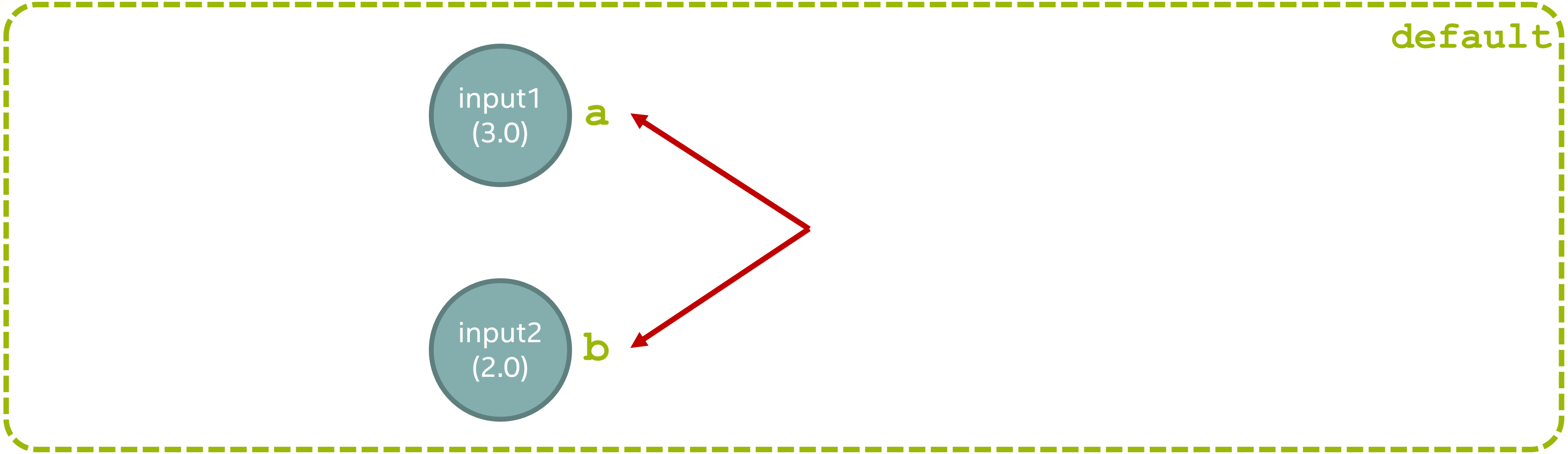
Additional Ops continue to populate the default graph

```
>>> b = tf.constant(2.0, name="input2")
```



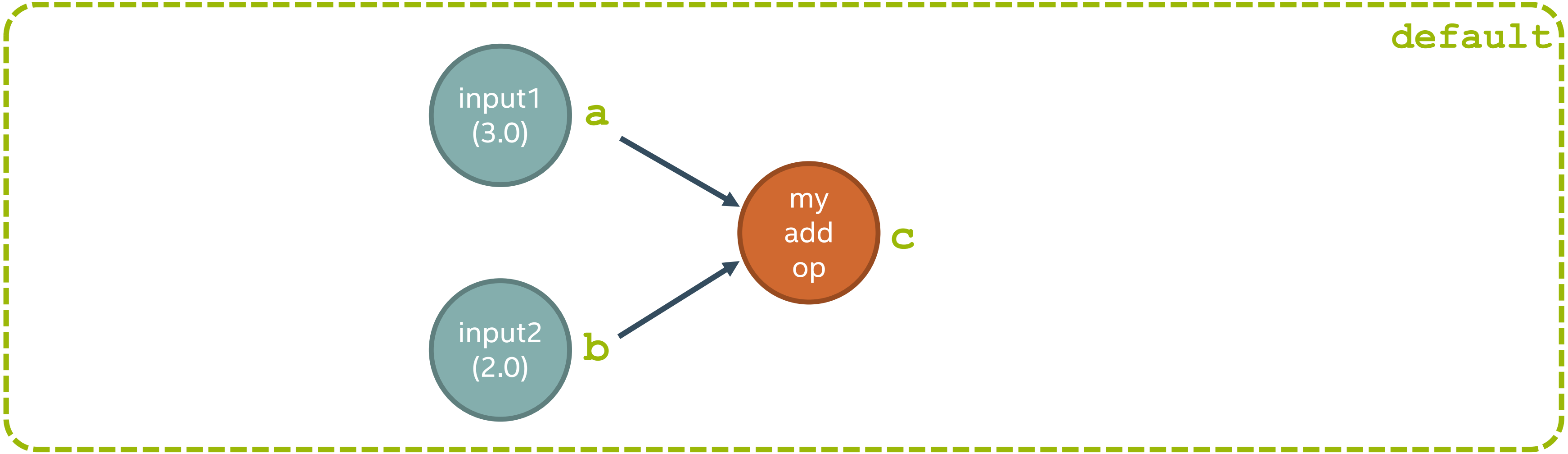
We have Python variable names which refer to the `Tensor` output of our two operations

```
>>> b = tf.constant(2.0, name="input2")
```



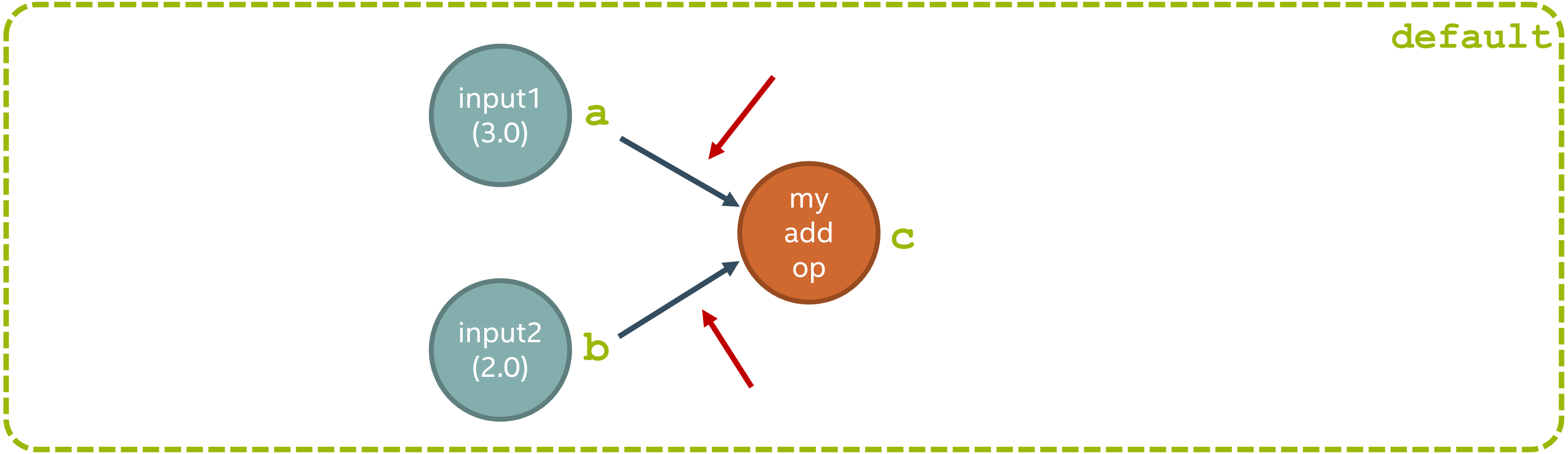
You can pass `Tensor` objects into other Ops

```
>>> c = tf.add(a, b, name="my_add_op")
```



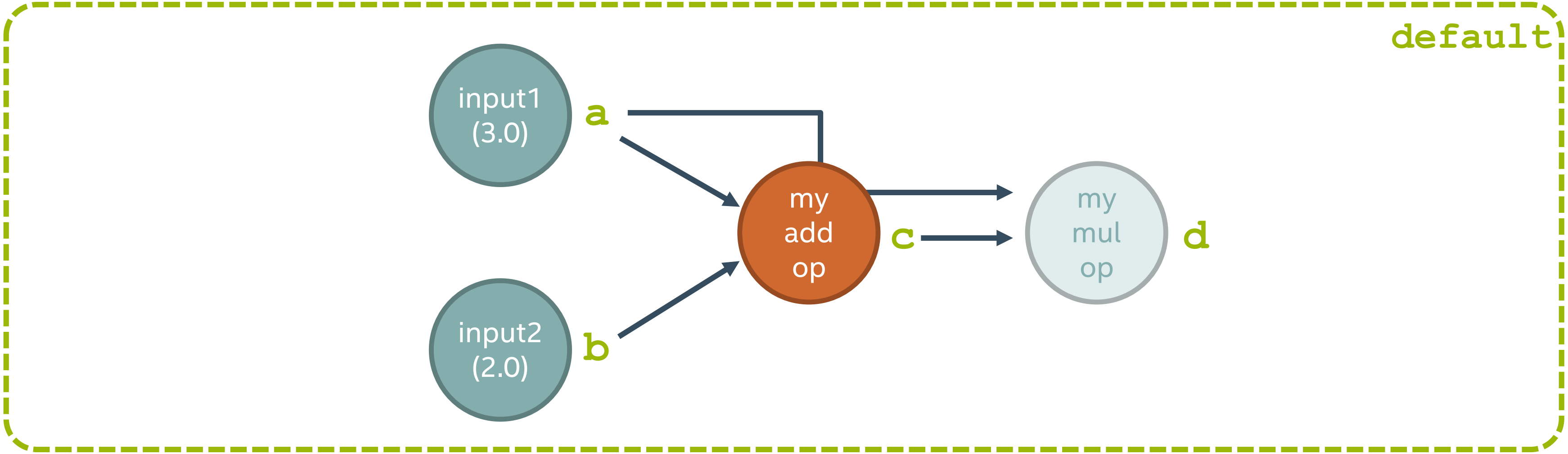
TensorFlow connects the referred Operations

```
>>> c = tf.add(a, b, name="my_add_op")
```



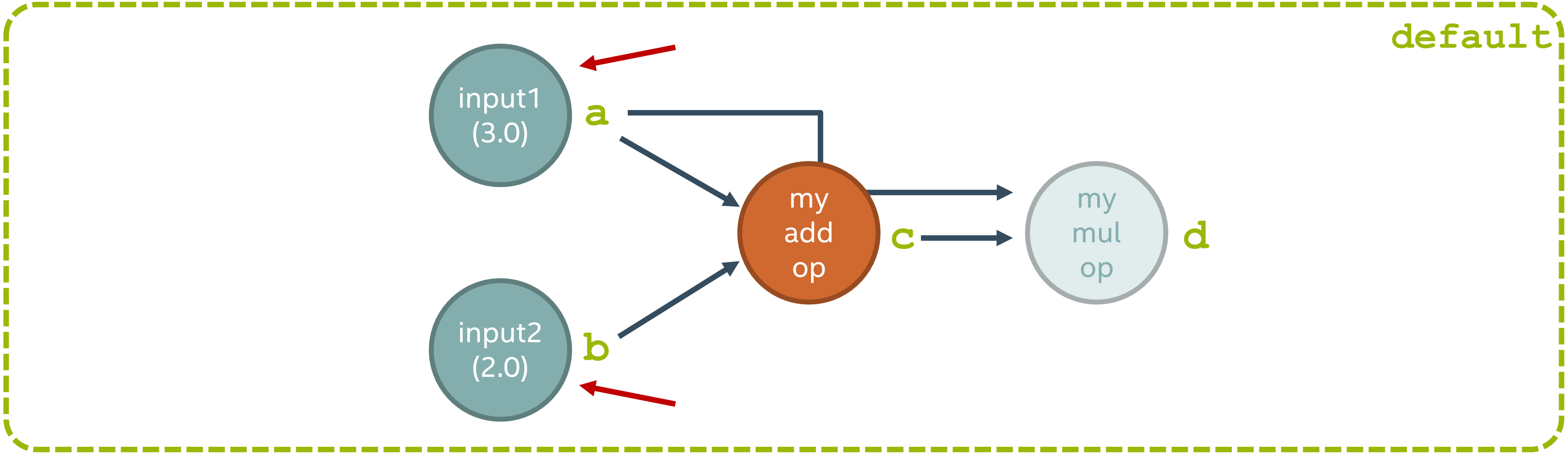
You can reuse the outputs of different Ops as much as you'd like

```
>>> c = tf.mul(a, c, name="my_mul_op")
```



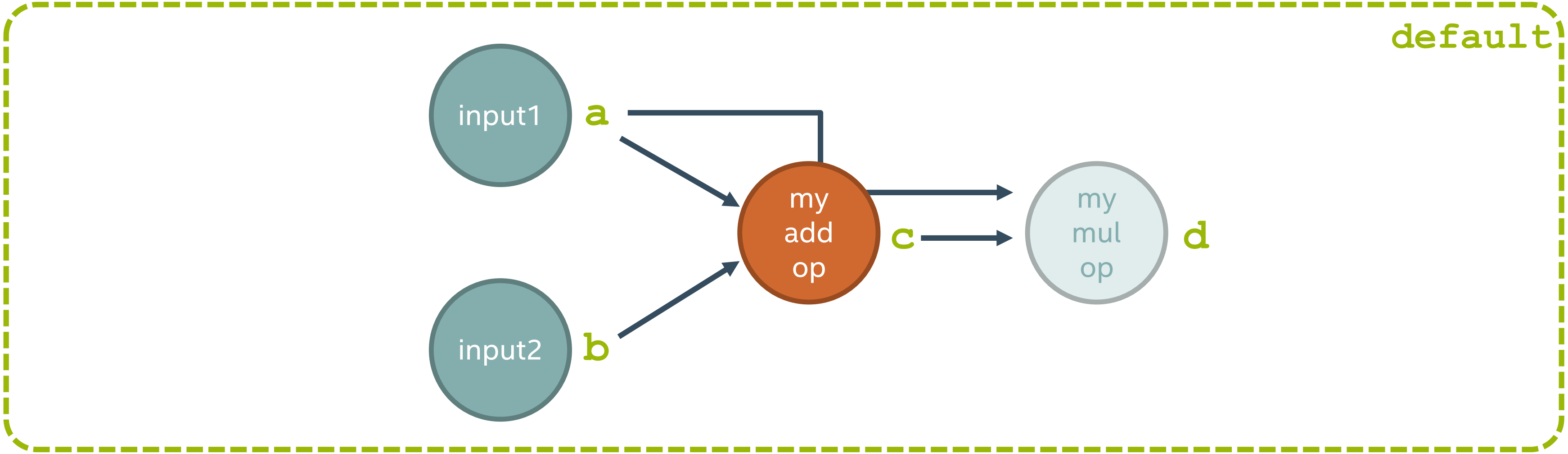
Recall: our inputs are fixed values (constant Ops)

```
>>>
```



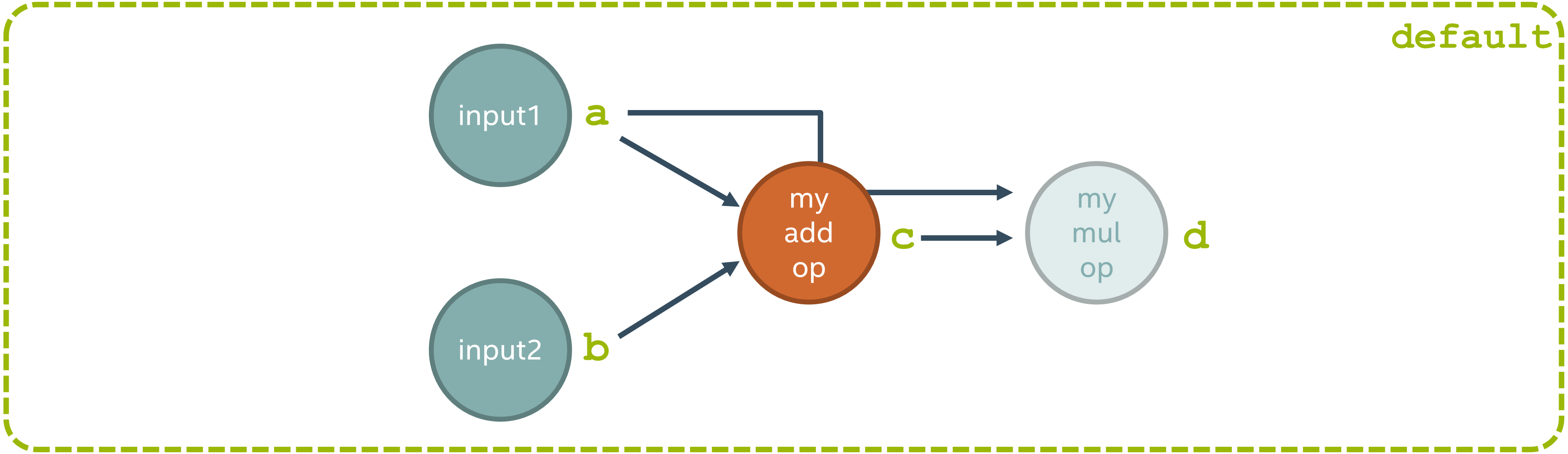
We could have used `tf.placeholder()` to define explicit input that vary run-to-run

```
>>> a = tf.placeholder(tf.float32, name="input1")
```



Side note: we're passing in the data type `tf.float32` to `tf.placeholder`

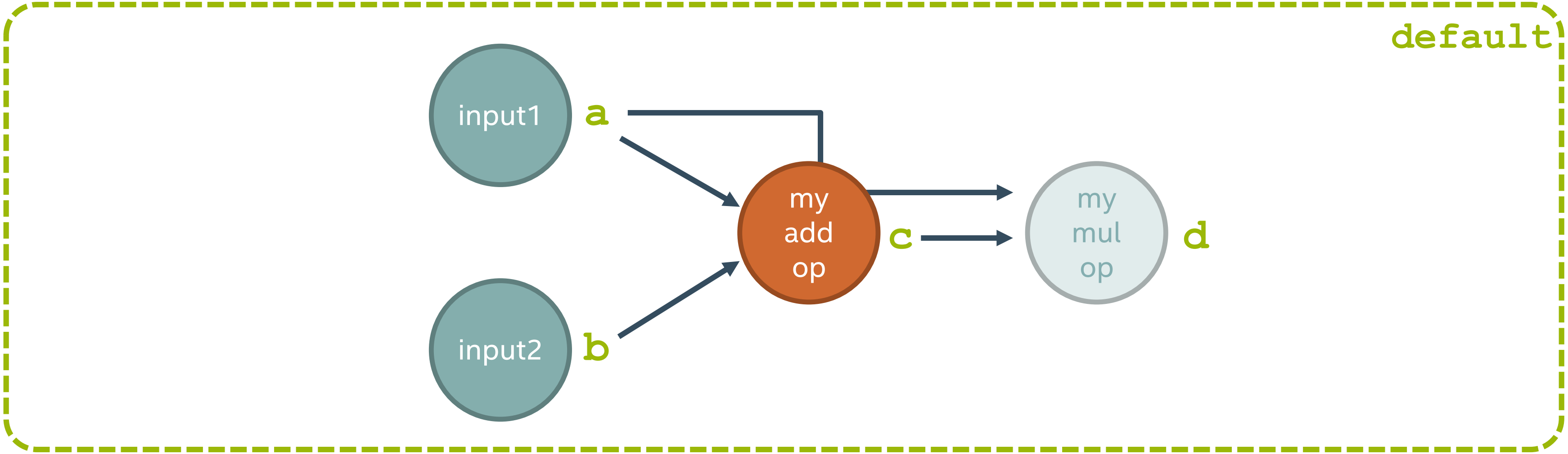
```
>>> a = tf.placeholder(tf.float32, name="input1")
```



RUNNING GRAPHS

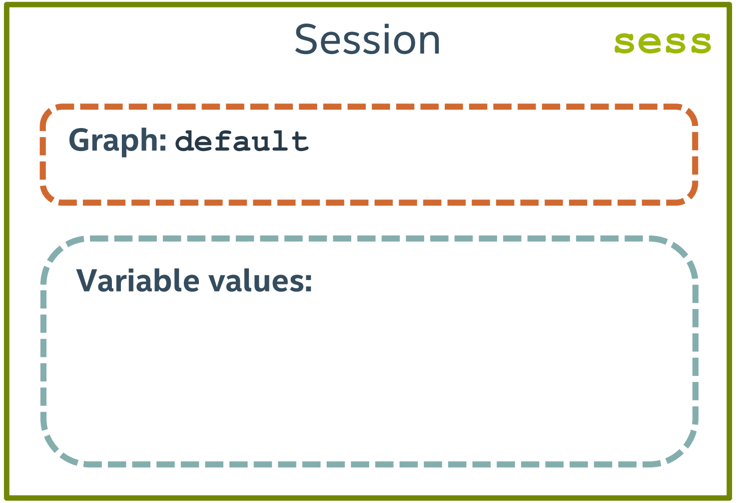
Now that we've created a graph, let's run it!

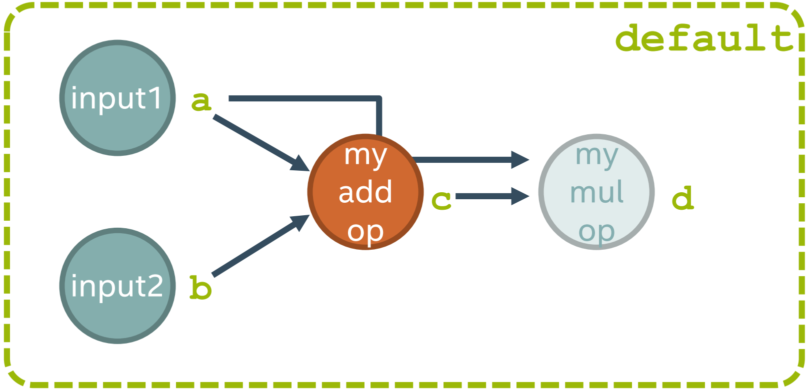
```
>>>
```



We use a `Session` object to execute graphs.
Each `Session` is dedicated to a single graph.

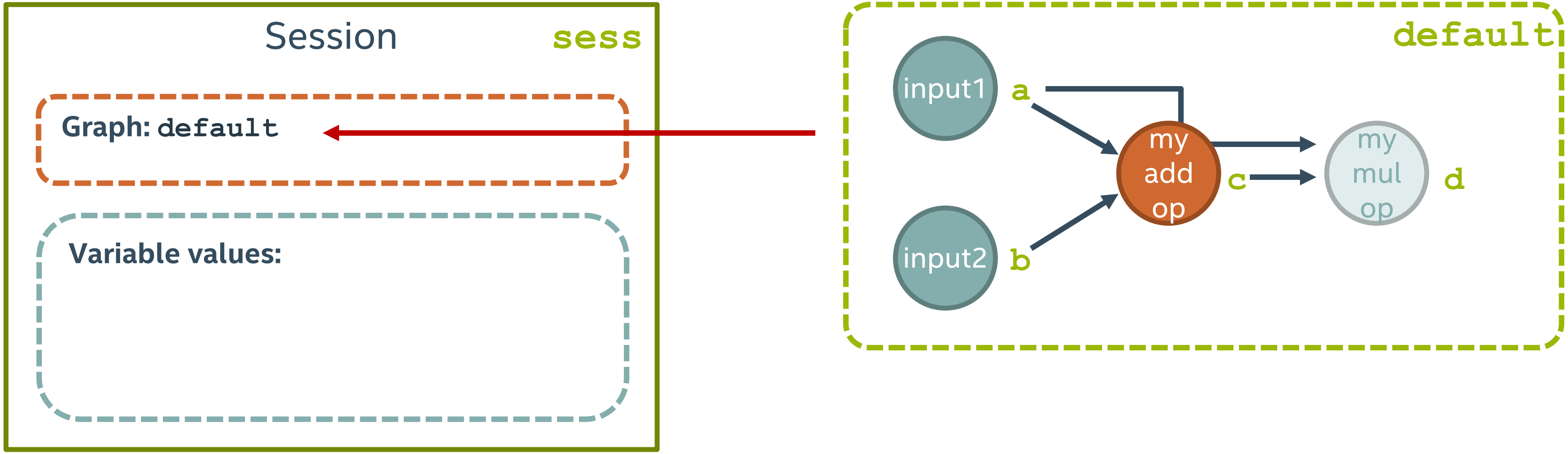
```
>>> sess = tf.Session()
```





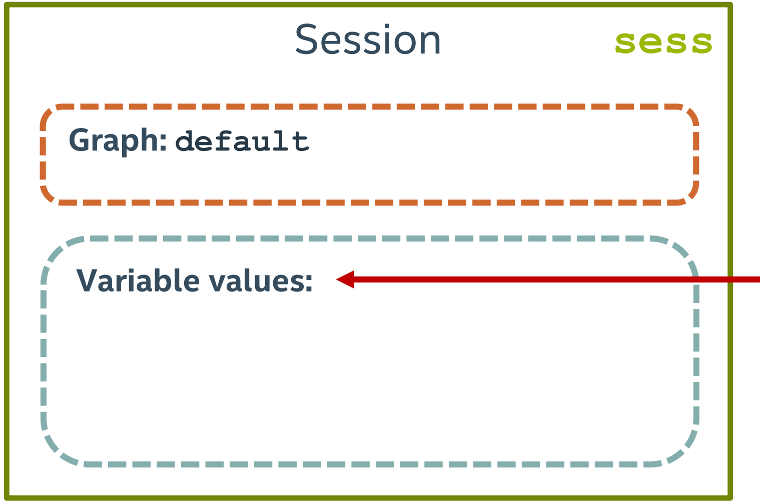
By default, a Session uses the current default graph

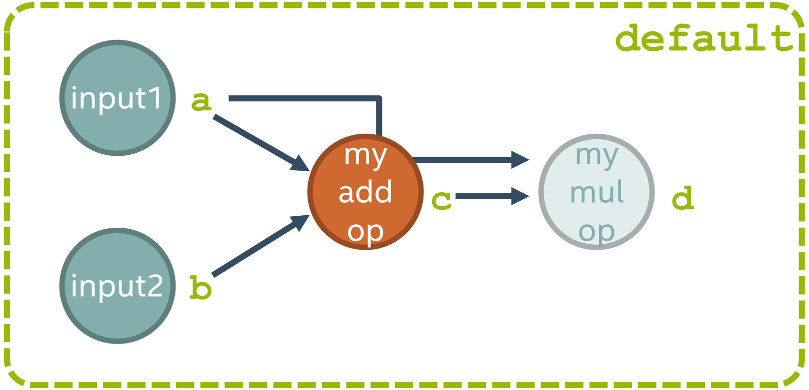
```
>>> sess = tf.Session()
```



We'll discuss Variables shortly.

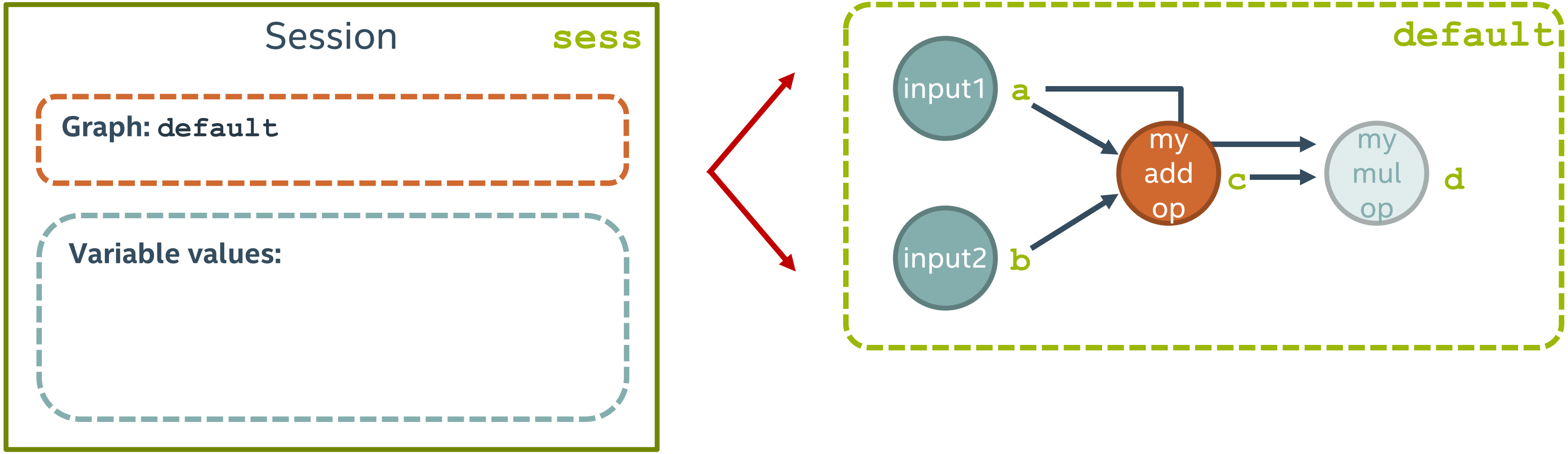
```
>>> sess = tf.Session()
```





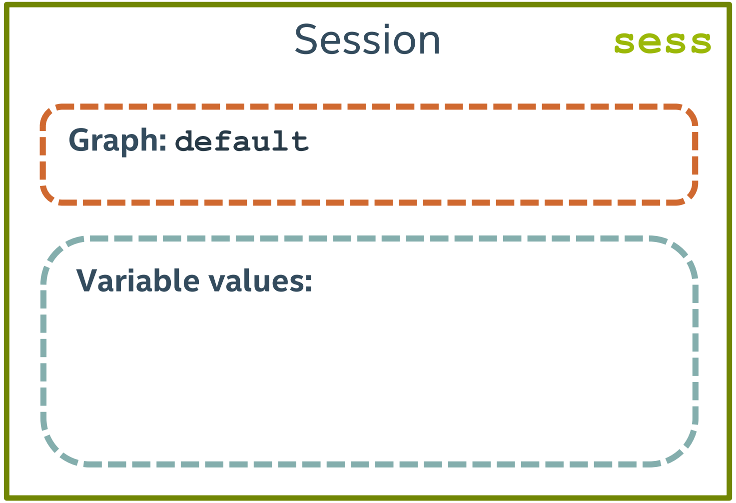
`placeholders` require data to fill them in when the graph is run

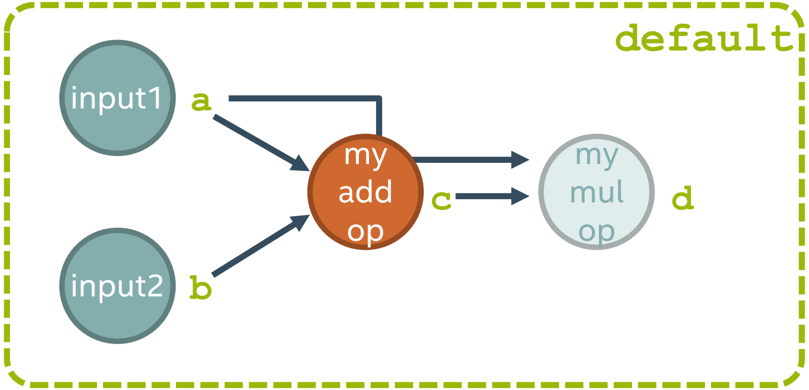
```
>>>
```



We do this by creating a dictionary mapping `Tensor` keys to numeric values

```
>>> feed_dict = {a: 3.0, b: 2.0}
```

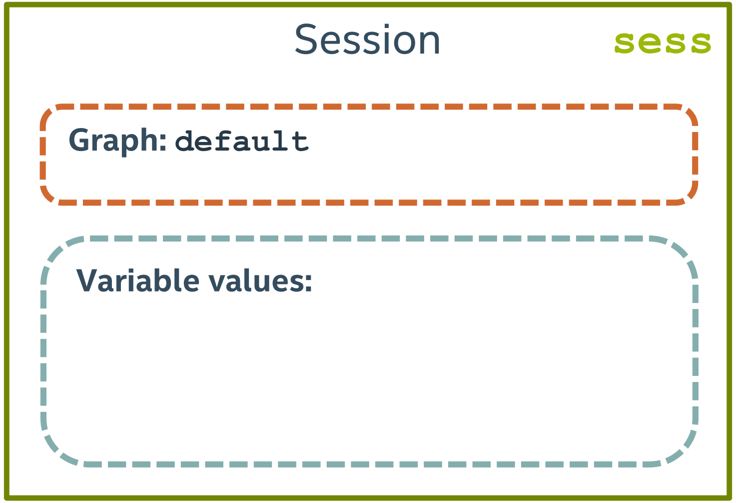


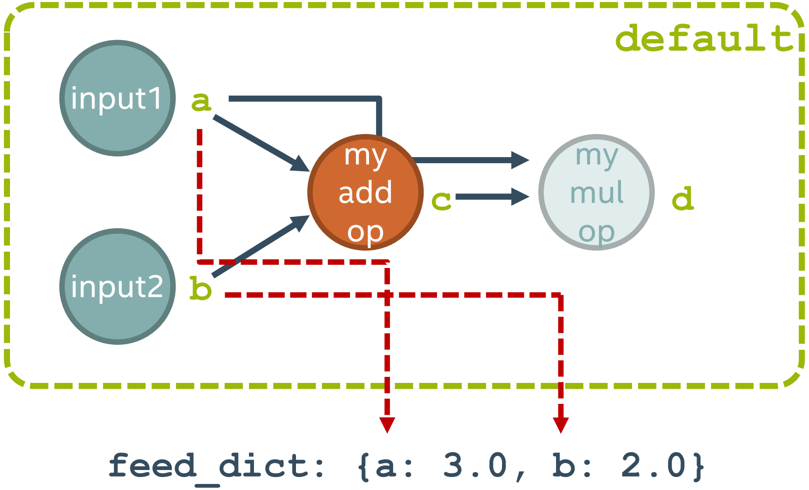



```
feed_dict: {a: 3.0, b: 2.0}
```

We pass in the same tensors as we used to create our graph

```
>>> feed_dict = {a: 3.0, b: 2.0}
```





We execute the graph with `sess.run(fetches, feed_dict)`

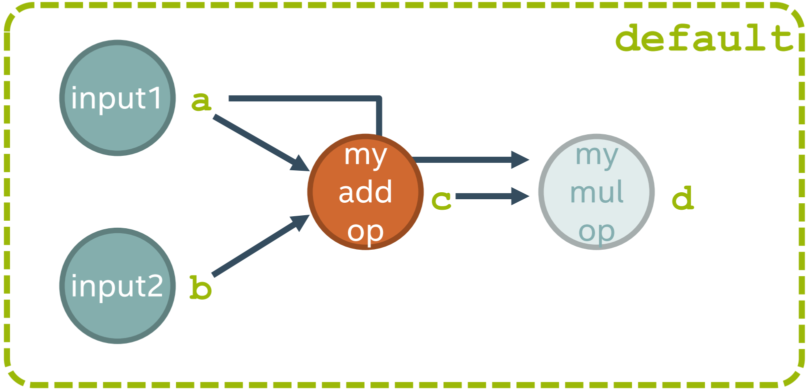
```
>>> out = sess.run(d, feed_dict=feed_dict)
```

Session **sess**

Graph: `default`

Variable values:

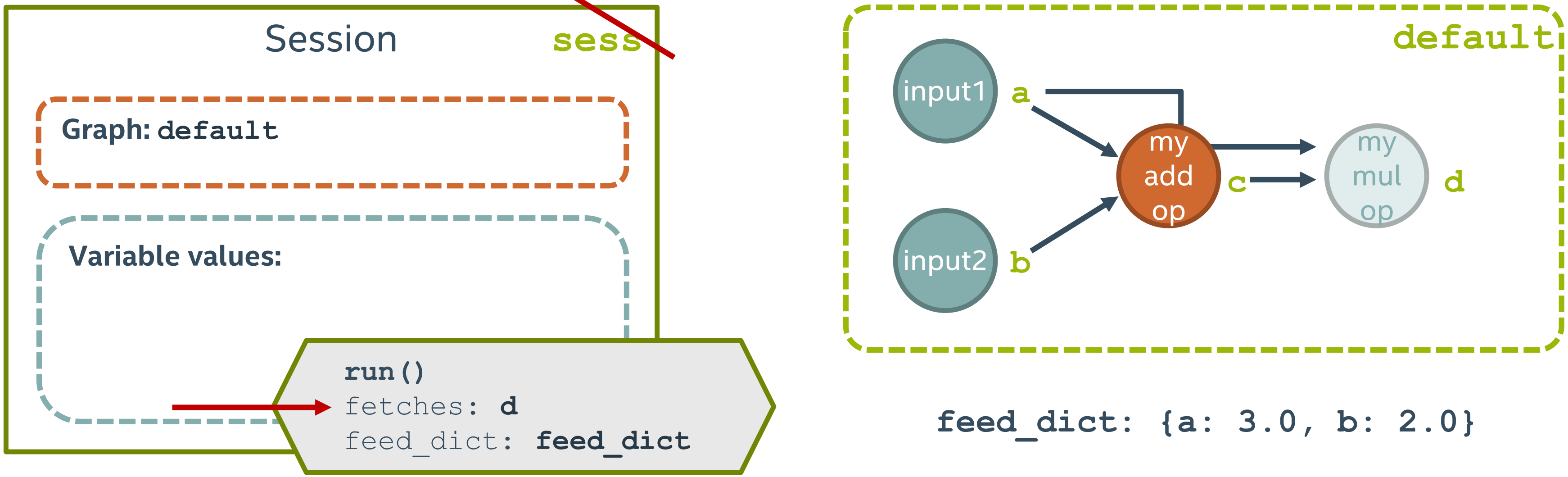
`run()`
`fetches: d`
`feed_dict: feed_dict`



`feed_dict: {a: 3.0, b: 2.0}`

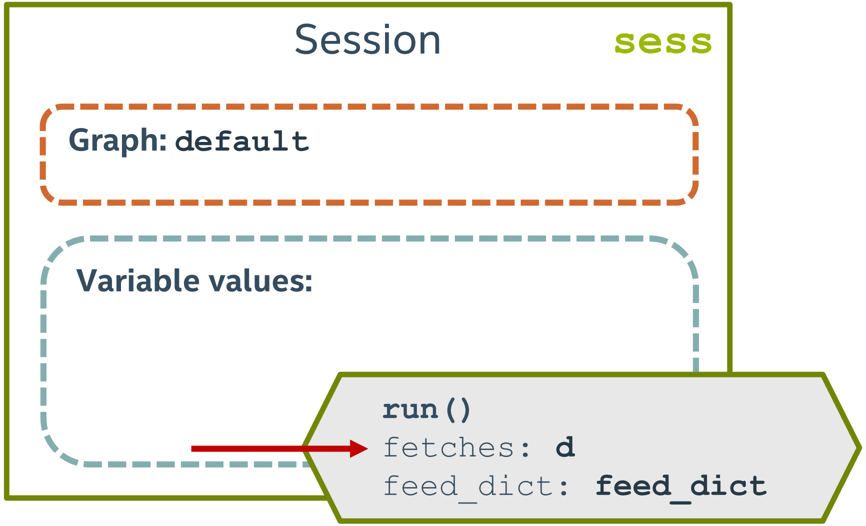
`Fetches` (a `Tensor`, `Operation`, or list of these) tells TensorFlow what outputs we want

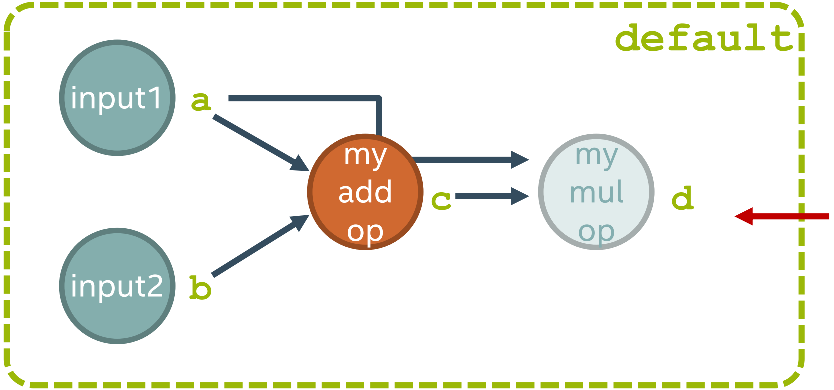
```
>>> out = sess.run(d, feed_dict=feed_dict)
```



Here, we request `d`, the `Tensor` output of my_mul_op

```
>>> out = sess.run(d, feed_dict=feed_dict)
```

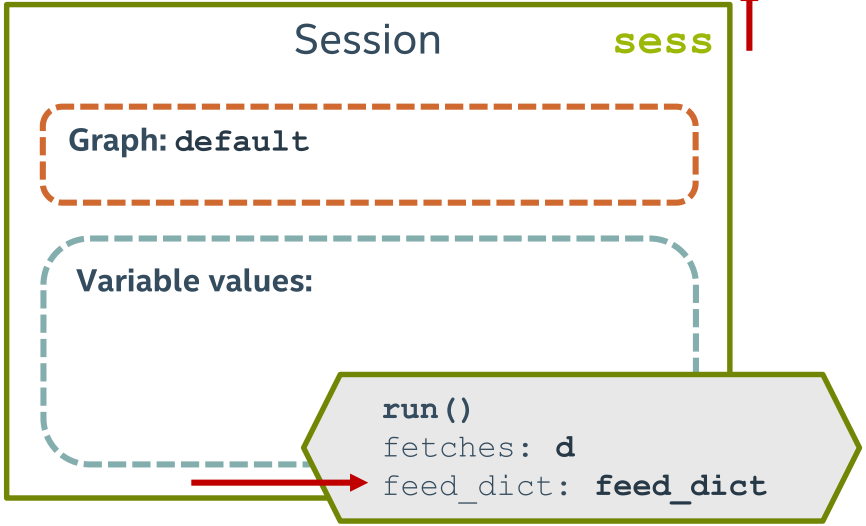


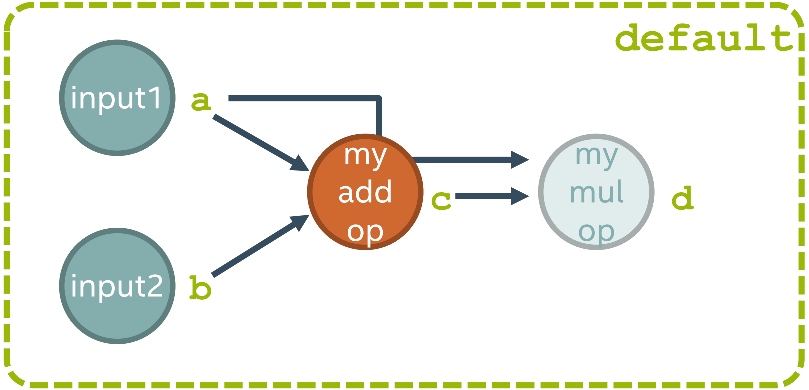


feed_dict: {a: 3.0, b: 2.0}

`feed_dict` tells the Session tensor substitutions

```
>>> out = sess.run(d, feed_dict=feed_dict)
```

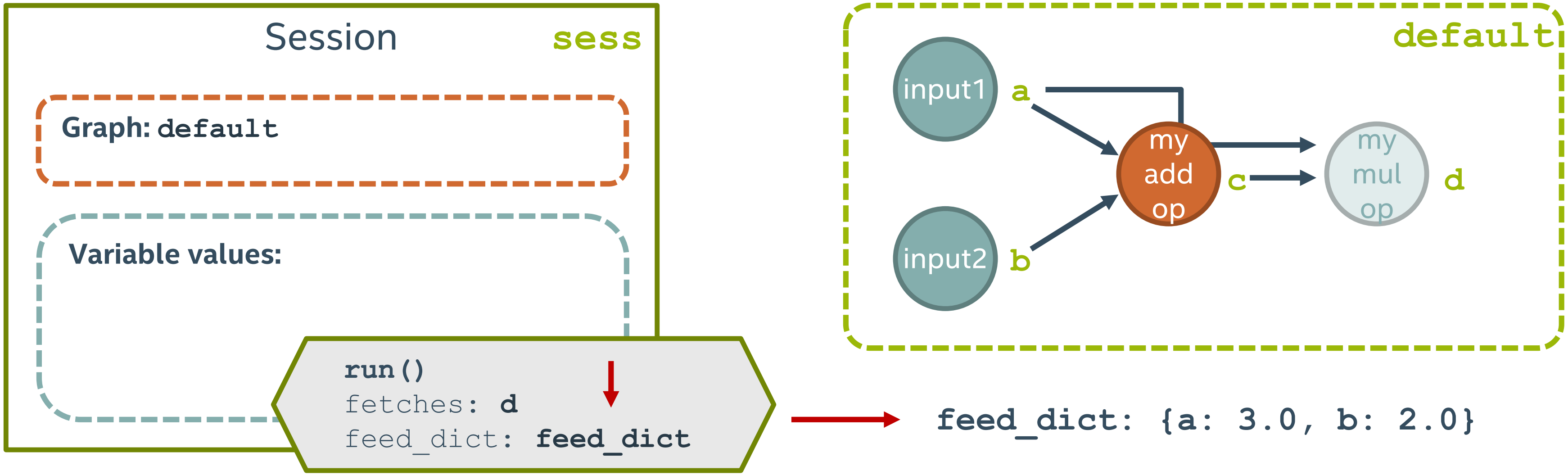





```
feed_dict: {a: 3.0, b: 2.0}
```

We created a `feed_dict`, in advance, which tells the Session to use 3.0 and 2.0 for `a` and `b`

```
>>> out = sess.run(d, feed_dict=feed_dict)
```



`placeholder` Ops *must be* feed a value

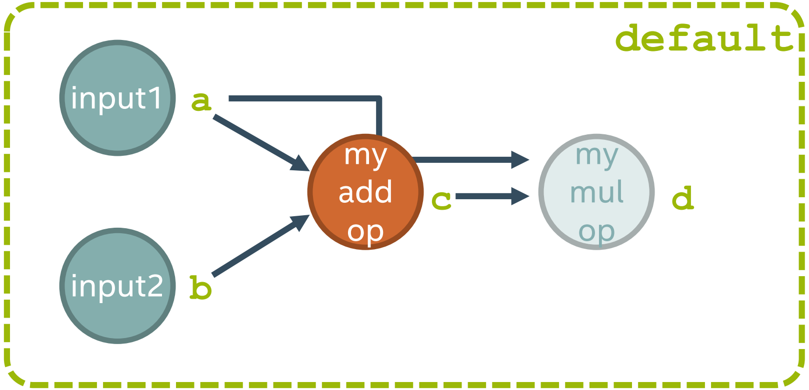
```
>>> out = sess.run(d, feed_dict=feed_dict)
```

Session **sess**

Graph: `default`

Variable values:

```
run()
fetches: d
feed_dict: feed_dict
```



`feed_dict: {a: 3.0, b: 2.0}`

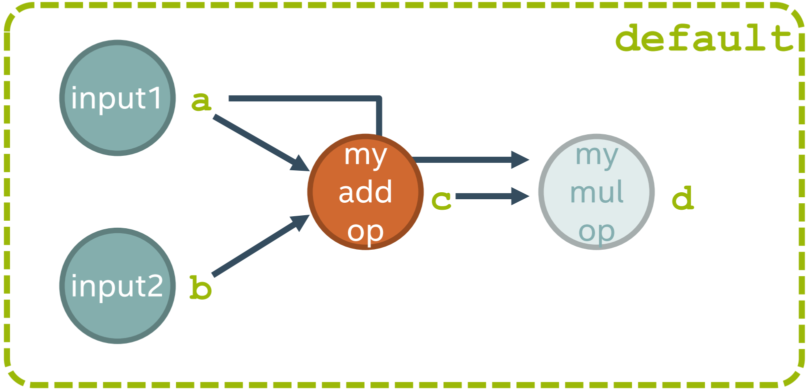
`sess.run` returns the fetched values as a NumPy array

```
>>>
```

Session `sess`

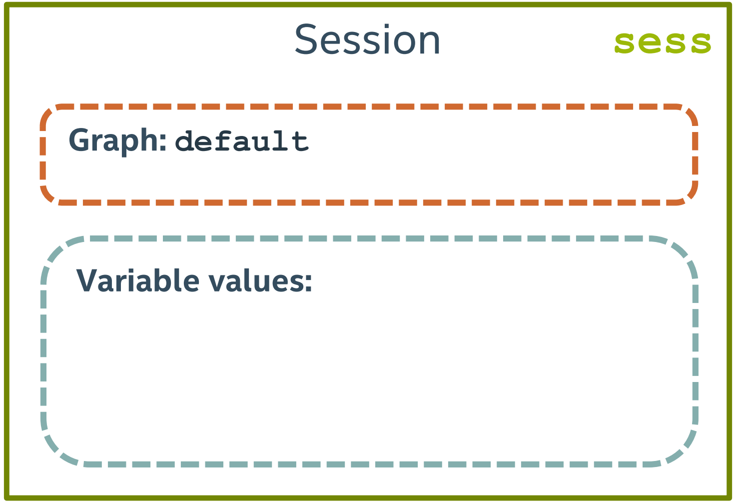
Graph: `default`

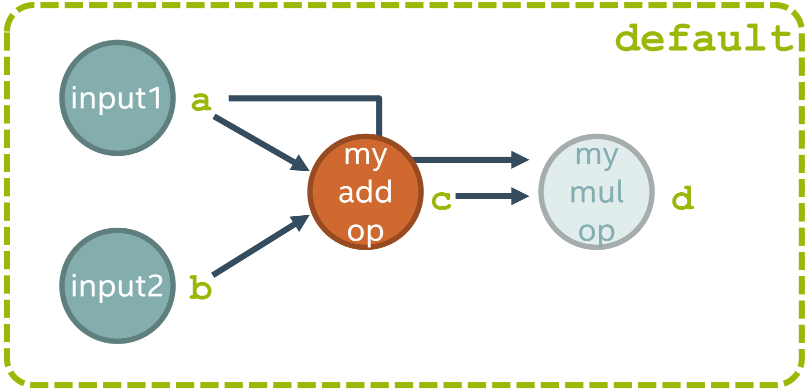
Variable values:




```
feed_dict: {a: 3.0, b: 2.0}
out = 15
```

```
>>> print(out)  # prints the value 15
```






```
feed_dict: {a: 3.0, b: 2.0}
out = 15
```

VARIABLES

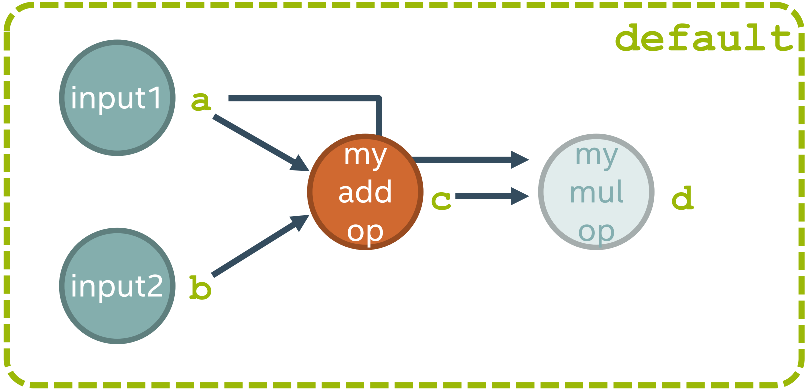
Our basic graph is well and good, but it would be nice to have stateful information. We can use `Variable` objects to accomplish this.

```
>>>
```

Session **sess**

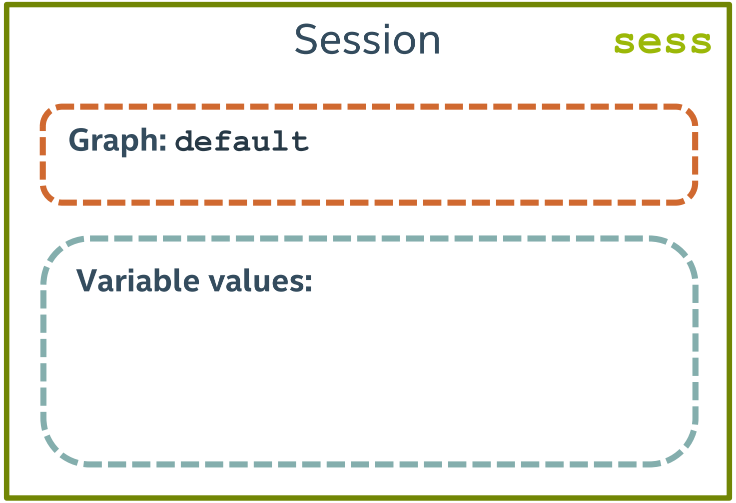
Graph: `default`

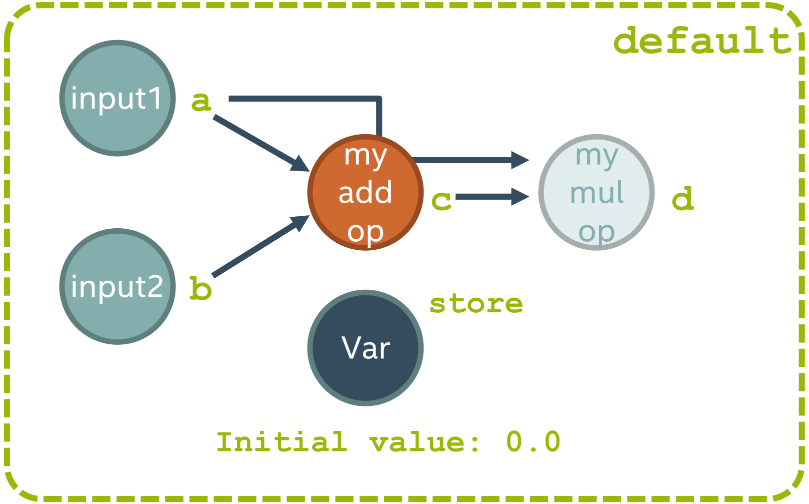
Variable values:



Our basic graph is well and good, but it would be nice to have stateful information. We can use `Variable` objects to accomplish this.

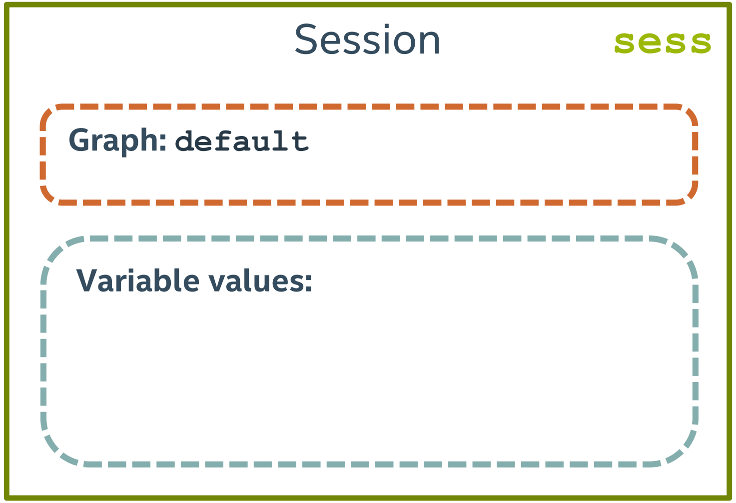
```
>>> store = tf.Variable(0.0, name="var")
```

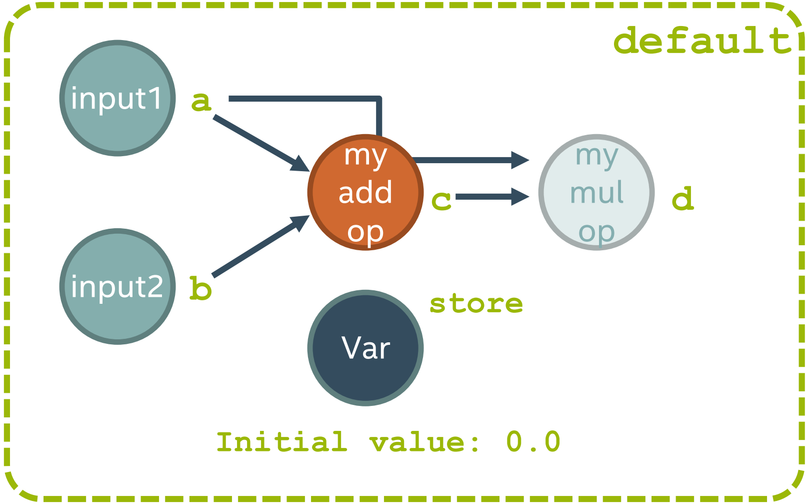




We need to pass in a initial value for our `Variable`, and we'll also give it a name.

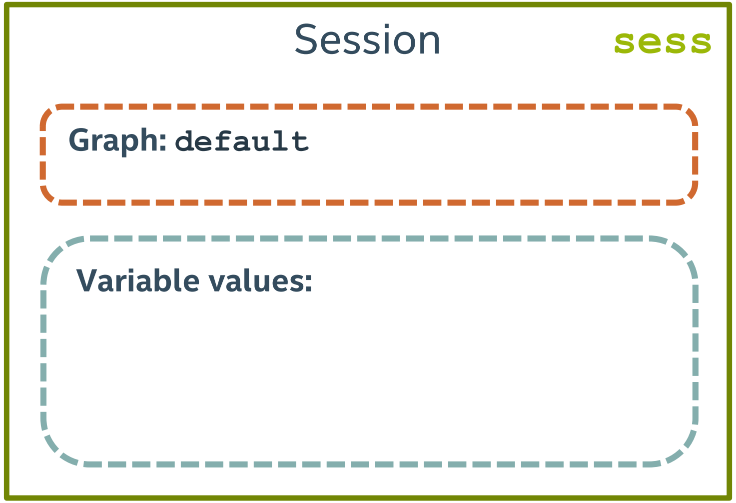
```
>>> store = tf.Variable(0.0, name="var")
```

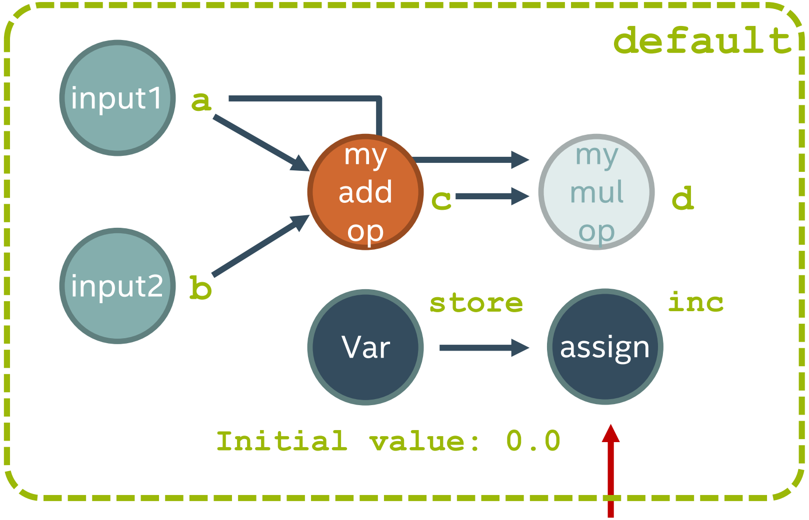




Let's also create an operation that allows us to increment the variable by 1

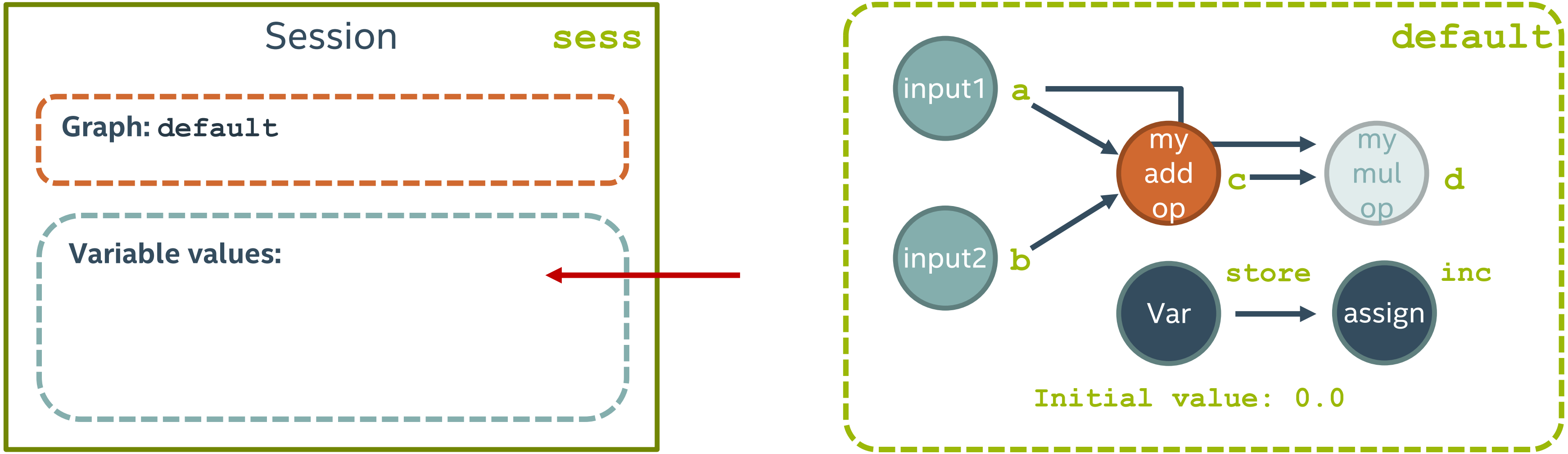
```
>>> inc = store.assign(store + 1)
```





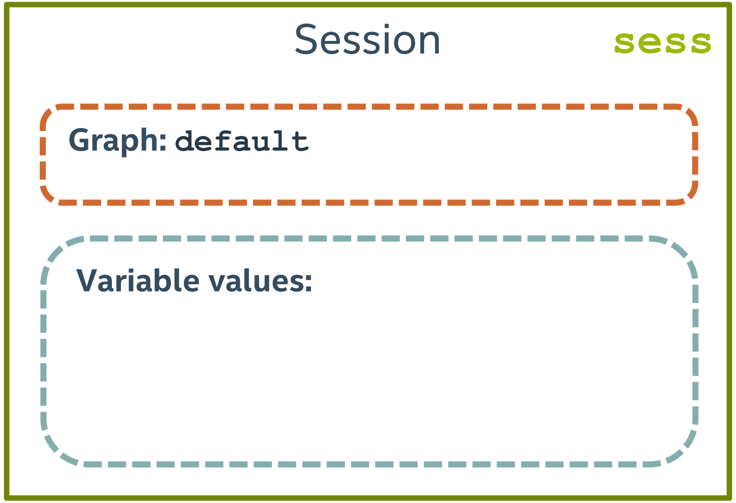
Note that our Session doesn't have any value for our `Variable` yet.

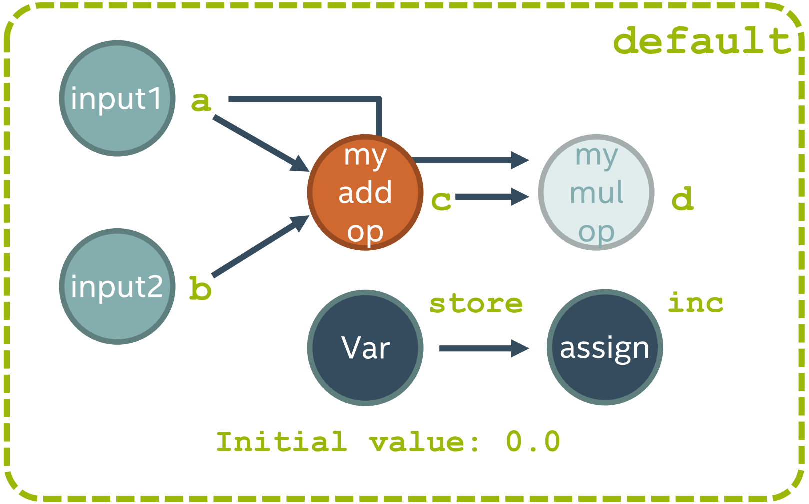
```
>>>
```



For our `Session` to use the `Variable`, we have to *initialize* it.

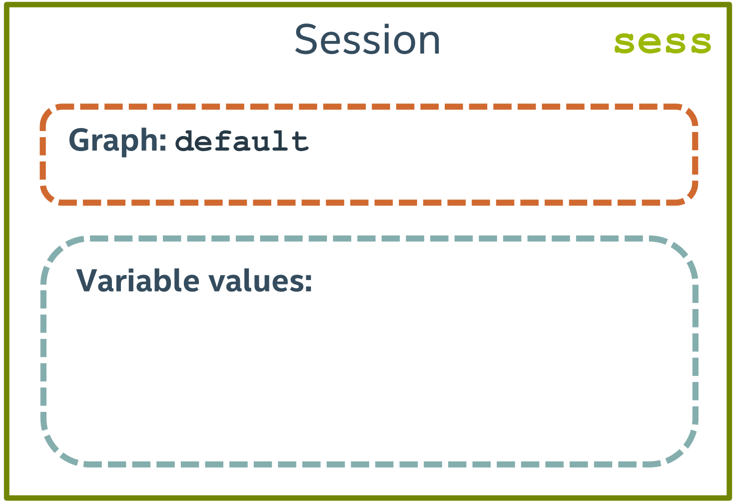
```
>>>
```

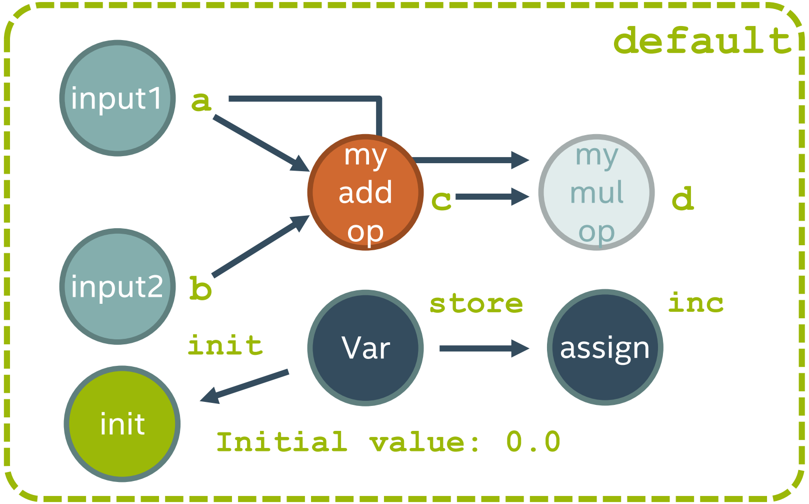




First, we'll create an initialization Op with `tf.global_variables_initializer()`

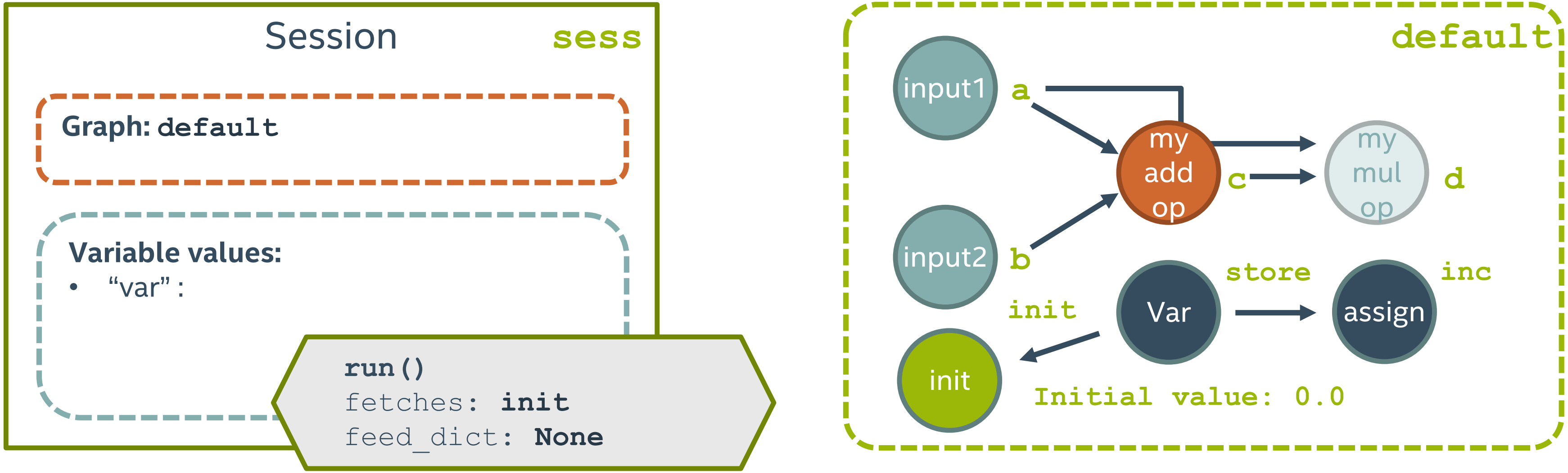
```
>>> init = tf.global_variables_initializer()
```





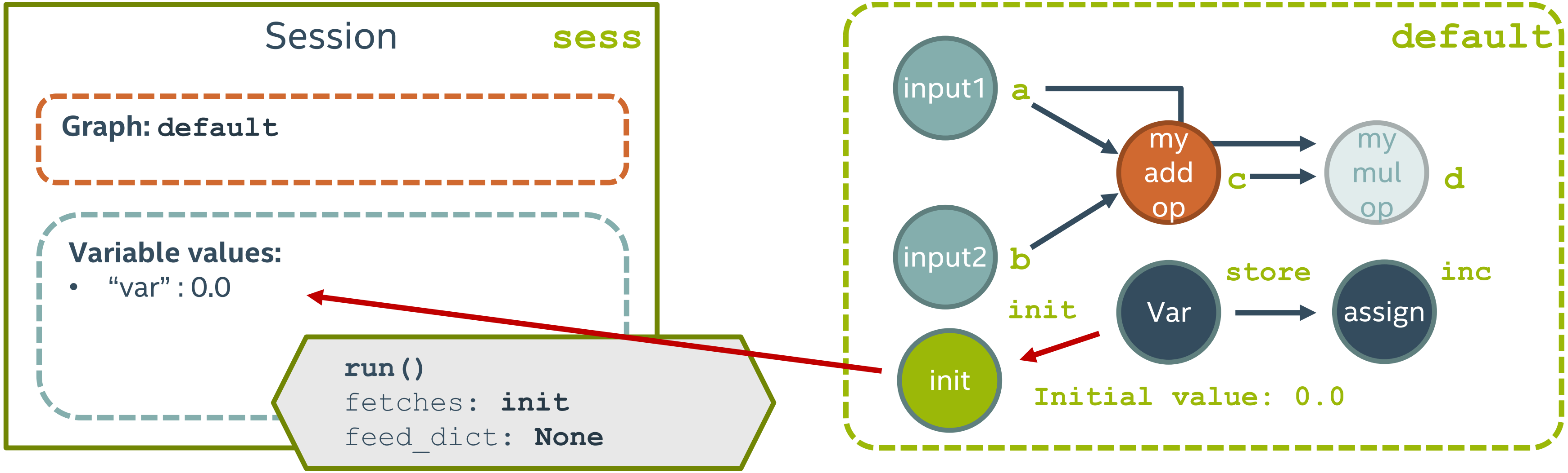
Then run the `init` Op in the `Session` we want to use the variables

```
>>> sess.run(init)
```



`init` reads the initial value from the variable and assigns it to the Session's store

```
>>> sess.run(init)
```



Now the `Session` will persist that value across multiple runs

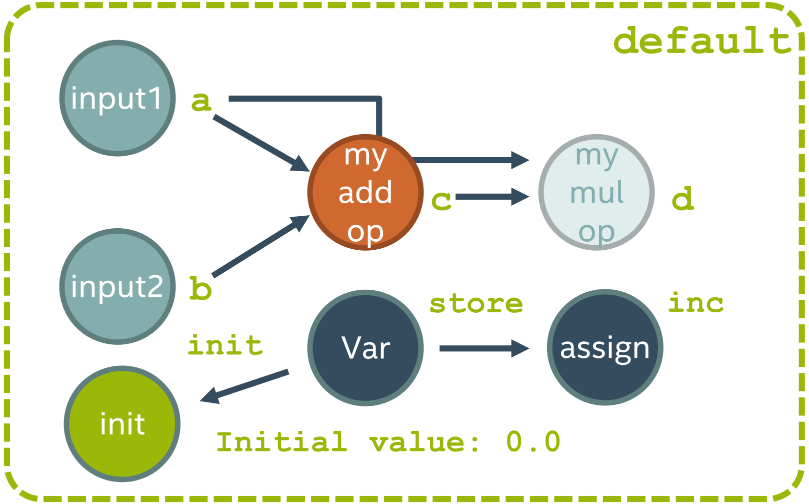
```
>>>
```

Session `sess`

Graph: `default`

Variable values:

- "var" : 0.0



We can increment the value by one with our `inc` Op

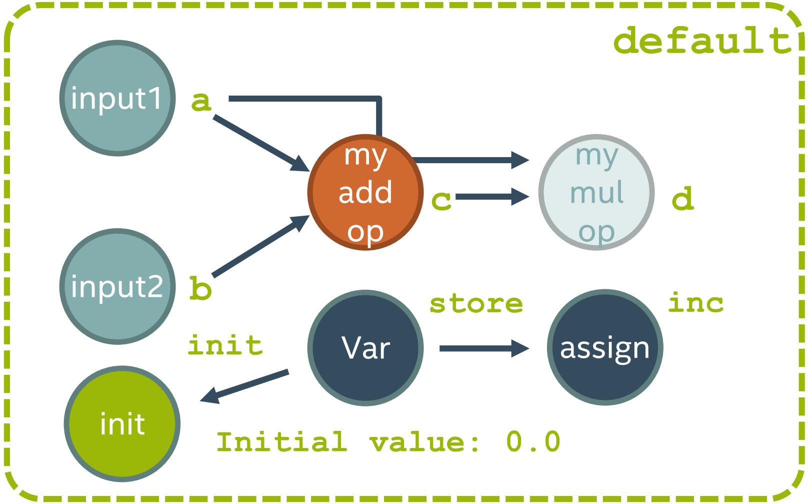
```
>>> sess.run(inc)
```

Session `sess`

Graph: `default`

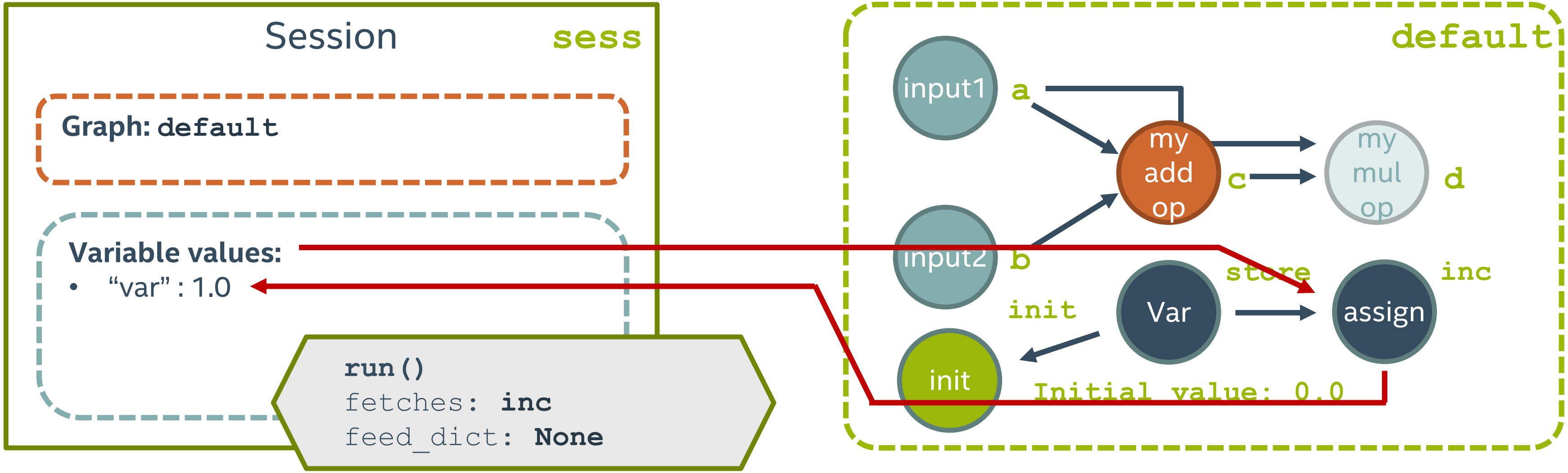
Variable values:
- "var" : 0.0

```
run()
fetches: inc
feed_dict: None
```



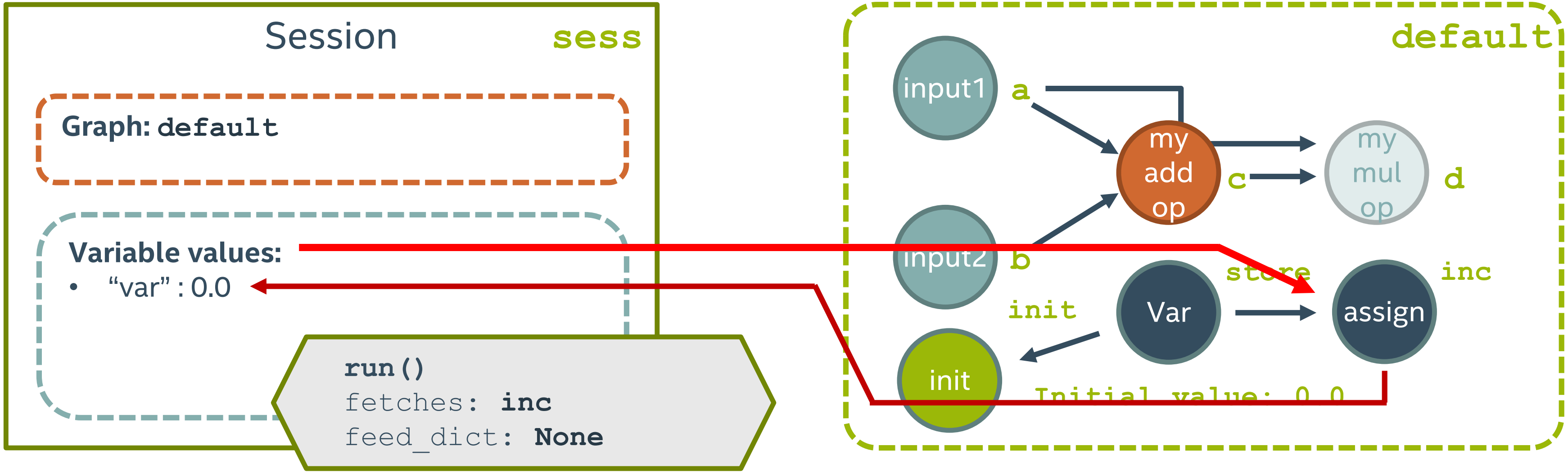
We can increment the value by one with `inc`

```
>>> sess.run(inc)
```



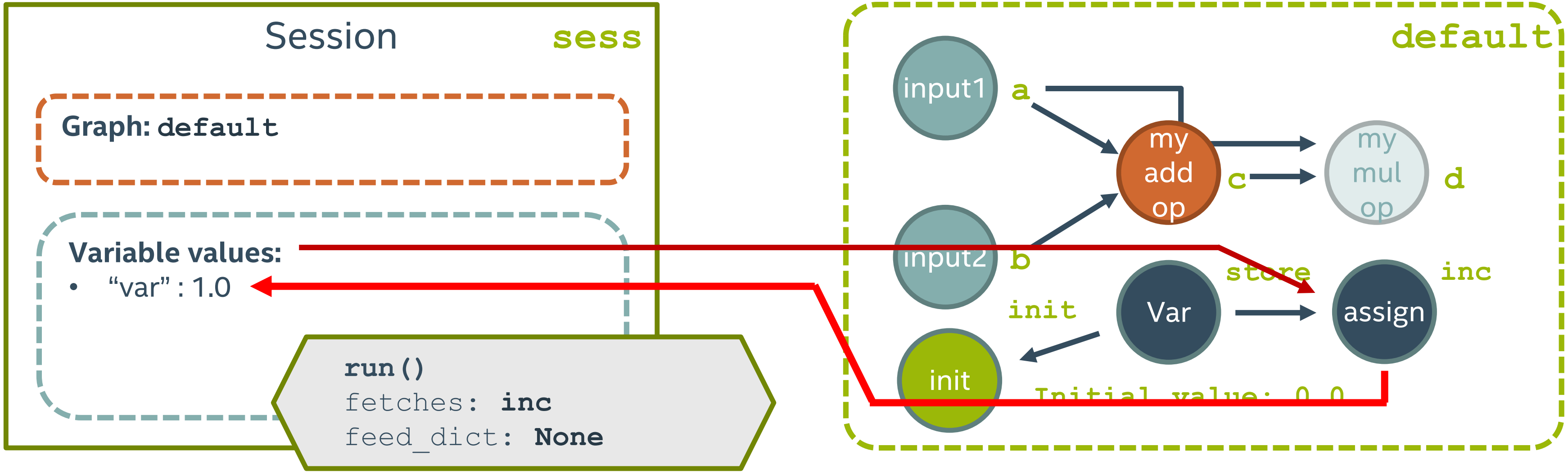
The assign `Operation` reads in the stored value from the `Session`

```
>>> sess.run(inc)
```



The Op calculates the new value (0 + 1) which is stored in the `Session`

```
>>> sess.run(inc)
```



Nervana™ AI Academy