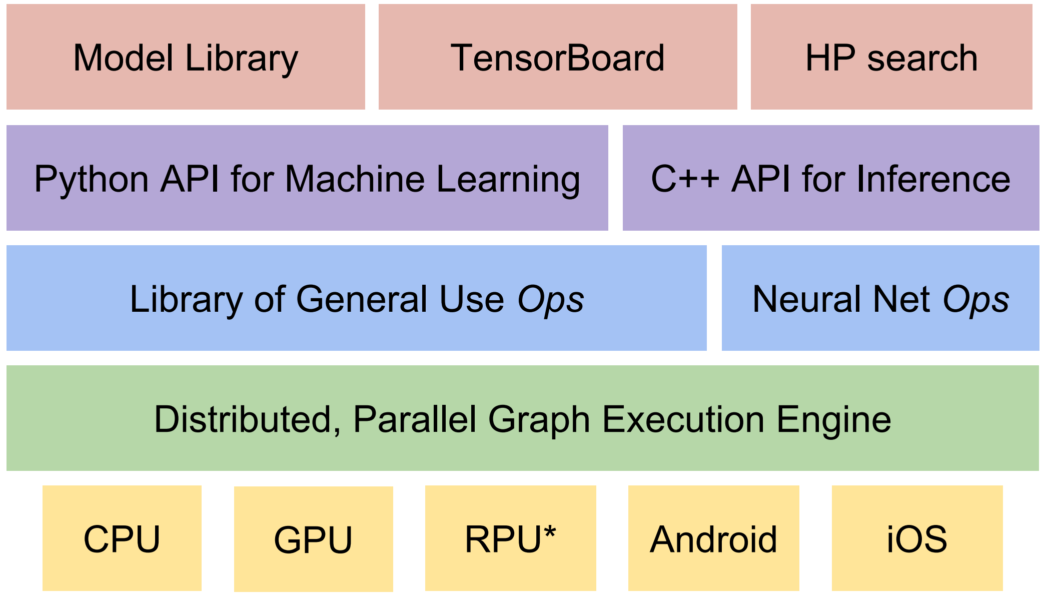
TensorFlow Queues

And Input Pipelines

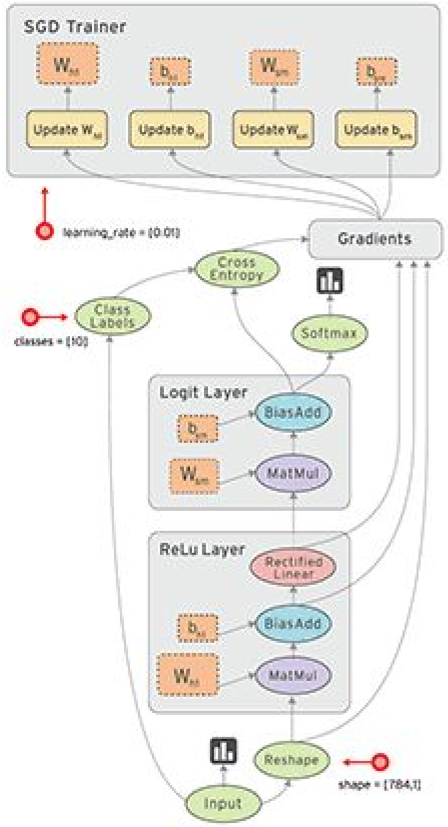
TensorFlow

- System for computation across heterogeneous devices
- Arbitrary operations, arbitrary devices, connected through C interface


Model Library
TensorBoard
HP search
Python API for Machine Learning
C++ API for Inference
Library of General Use *Ops*
Neural Net *Ops*
Distributed, Parallel Graph Execution Engine
CPU
GPU
RPU*
Android
iOS

Parallel Execution

- Launch graph in a *Session*
- Request output of some Ops with *Run* API
- TensorFlow computes set of Ops that must run to compute the requested outputs
- Ops execute, in parallel, as soon as their inputs are available

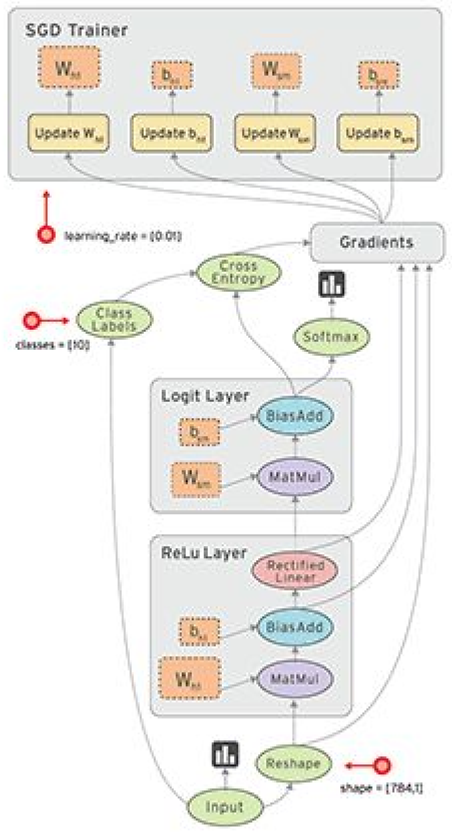


Parallel Execution with Python

1 core

```
for d in data:
   preprocess(d)
Python_code
Python_code
Python_code
```

48 cores (2x Intel Xeon)
16 GPUs (8 x K80)



GIL!!!!!!

Hacks around GIL

Python-code

Py_BEGIN_ALLOW_THREADS

```
Python code
Python code
Numpy.complex_calculation
…
…

Python code
```

Py_END_ALLOW_THREADS

Getting your data into TensorFlow

1. tf.constant(mydata) BAD
2. session.run(node, feed_dict={in:data}) BETTER
3. Queues and Input Pipelines BEST!

tf.constant(data)

- Used in tensorflow/examples/how_tos/reading_data/fully_connected_preloaded.py

```
with tf.device('/cpu:0'):
    input_images = tf.constant(data_sets.train.images)
    input_labels = tf.constant(data_sets.train.labels)
```

- Inlines data in the Graph definition
- hard 2GB limit for size of Graph
- Single-threaded (Graph not thread-safe)
- Running remotely adds additional protobuf encoding/decoding step

sess.run(..., feed_dict=...)

- Gets pointer to underlying numpy memory buffer
- Single-threaded memcpy (<2GB/s max)

BUT:

- Could be replaced with multi-threaded memcpy in future
- Can do sess.run calls in parallel, sess.run releases GIL

sess.run(..., feed_dict=...)

- What if you need to preprocess the data?

```
im = random_crop(im)
im = random_flip_left_right(im)
im = random_brightness(im)
im = random_contrast(im)
im = per_image_whitening(im)
```

- What if you need to implement pre-fetching/buffering?
    - ...

sess.run(..., feed_dict=...)

- What if you need to preprocess the data?

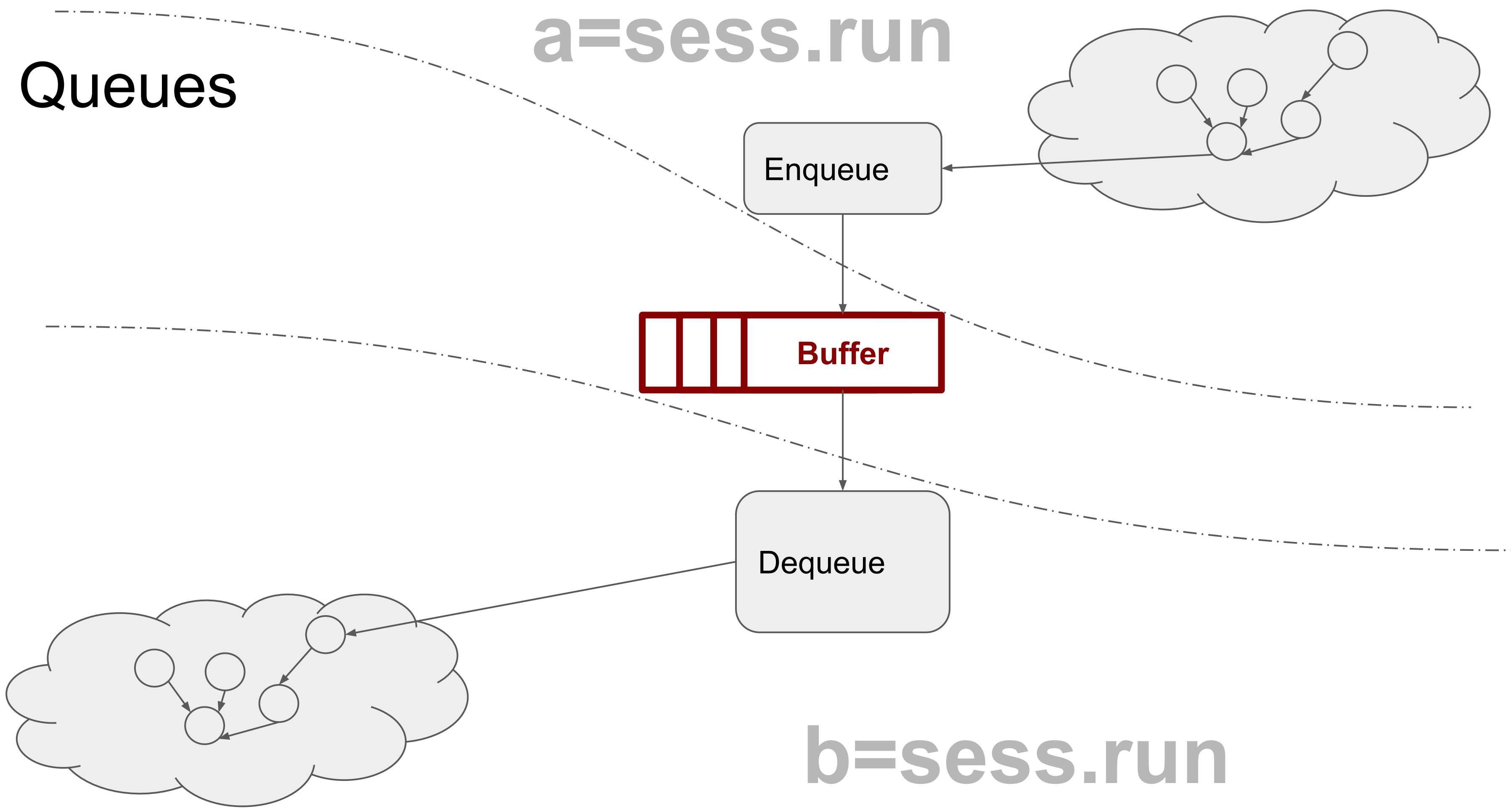
```
im = random_crop(im)
im = random_flip_left_right(im)
im = random_brightness(im)
im = random_contrast(im)
im = per_image_whitening(im)
```

$\longleftarrow$ TensorFlow native implementations

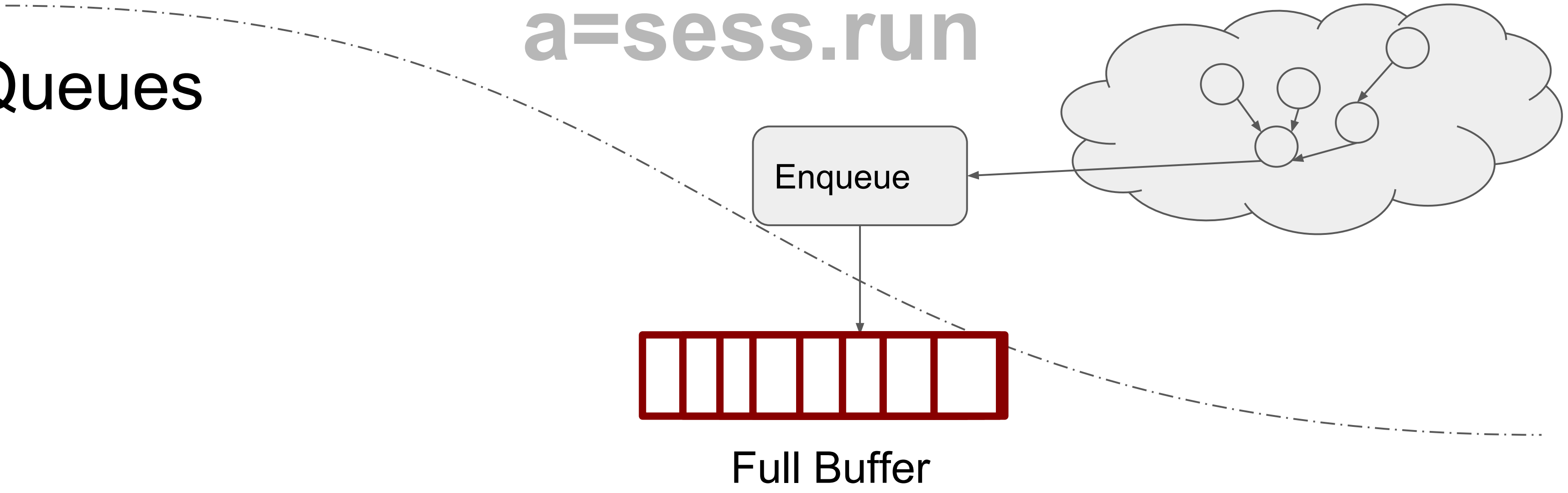
- What if you need to implement pre-fetching/buffering?
  - ….

$\longleftarrow$ Queues

Queues

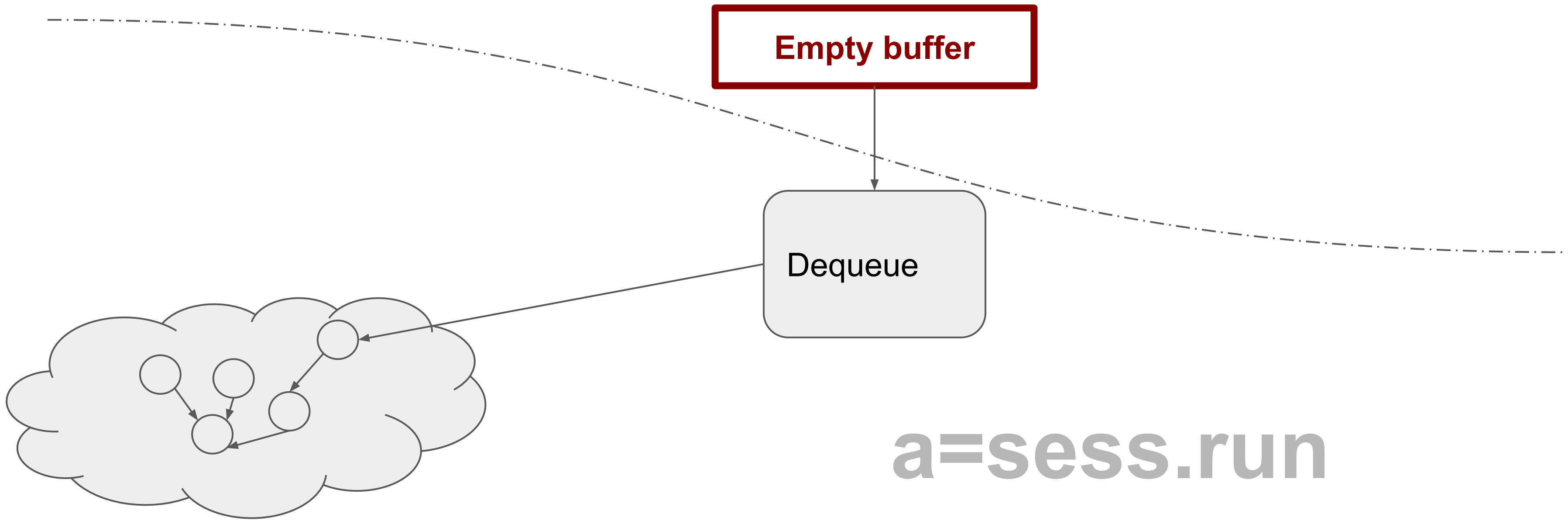


Queues



Hang forever

Hang forever



Hang forever unless

Session was configered with timeouts

session = tf.Session(tf.Config(operation_timeout_in_ms=2000))

DeadlineExceededError

OR

Queue was closed by pushing a special “Close” token on it.

sess.run(queue.close())

OutOfRangeError/AbortedError

Queue Example

```
q = tf.FIFOQueue(capacity=20, dtypes=[tf.int32])
enqueue_placeholder = tf.placeholder(tf.int32)
enqueue_op = q.enqueue(enqueue_placeholder)
sess = create_session()
for i in range(10):
    sess.run(enqueue_op, feed_dict={enqueue_placeholder:i})
    print "Queue size is now: "+str(sess.run(q.size()))
sess.run(q.close())
```

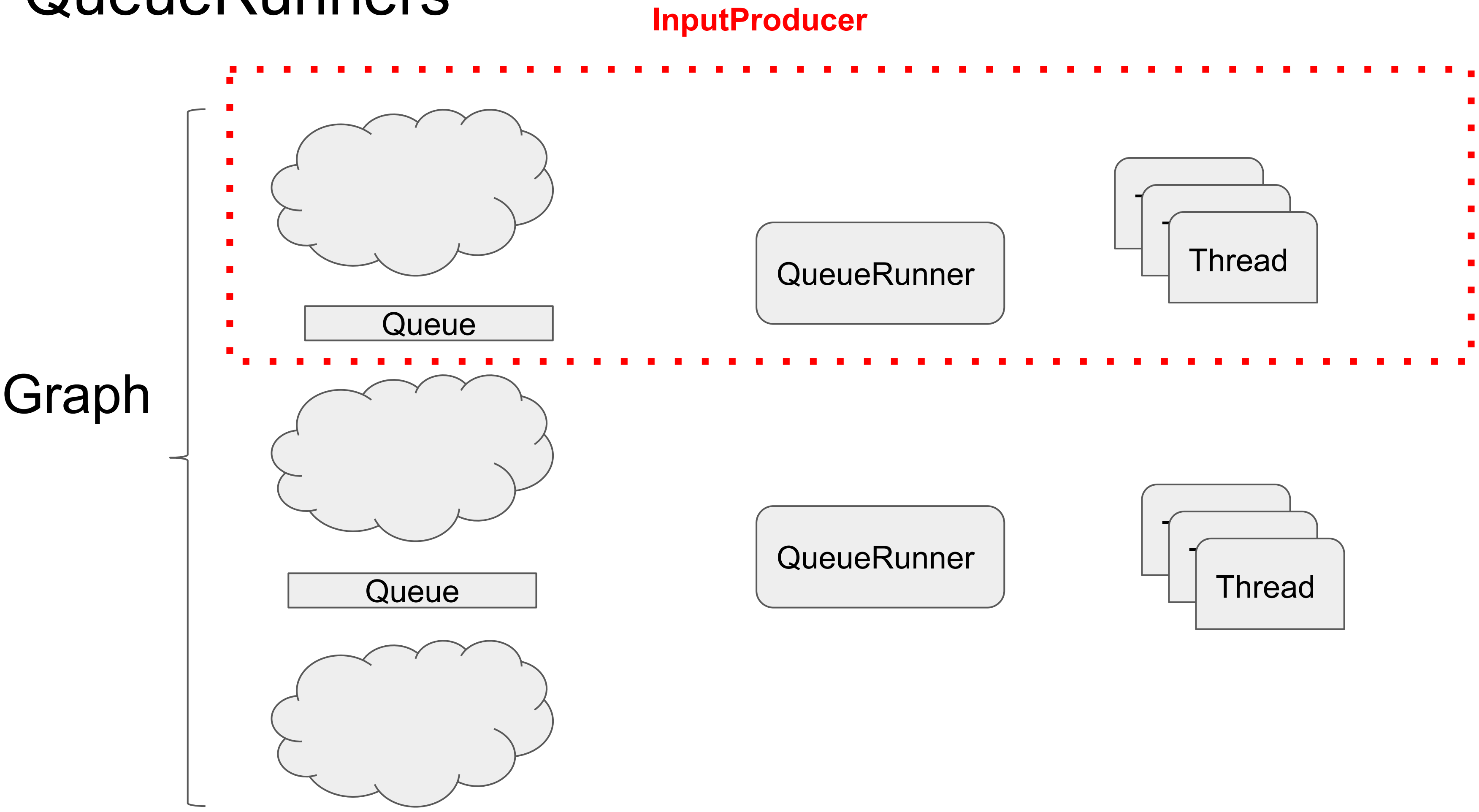
Queue Example

```
random_number = tf.random_uniform(shape=())
q = tf.FIFOQueue(capacity=20, dtypes=[tf.float32])
enqueue_op = q.enqueue(random_number)

sess = create_session()
print sess.run(q.size())   # prints 0
def run():
  for i in range(5):
    sess.run(enqueue_op)

threads = [threading.Thread(target=run, args=()) for i in range(2)]
[t.start() for t in threads]
print sess.run(q.size())    # prints 4
time.sleep(0.5)
print sess.run(q.size())    # prints 10
```

QueueRunners



Gotchas

- Queues are stateful

```
merged = make_image_input()
train_step = make_train_step(merged)
for step in range(5):
    loss_value = sess.run(train_step,feed_dict={y_label:y_1})
    summary_str = sess.run(merged,feed_dict={y_label:y_1})
    writer.add_summary(summary_str,step)

# where did extra images go?
```

Gotchas

```
ranges = tf.train.range_input_producer()
number = ranges.dequeue()
sess = create_session
tf.train.start_queue_runners()

if version==1:
  print sess.run([number]*3)
elif version ==2:
  print sess.run([ranges.dequeue()]*3)
elif version == 3:
  print sess.run([ranges.dequeue(), ranges.dequeue(), ranges.dequeue()])
```