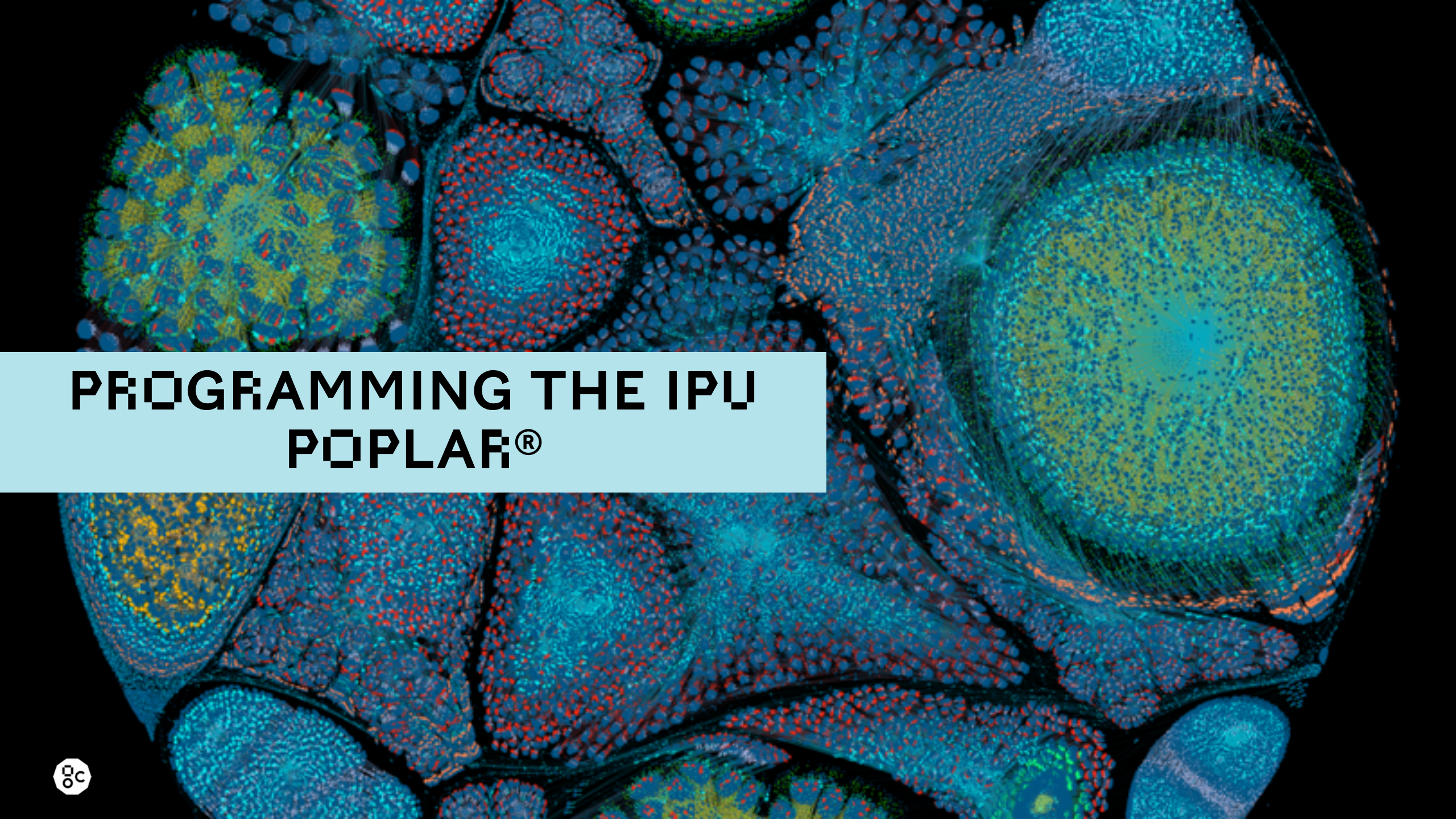
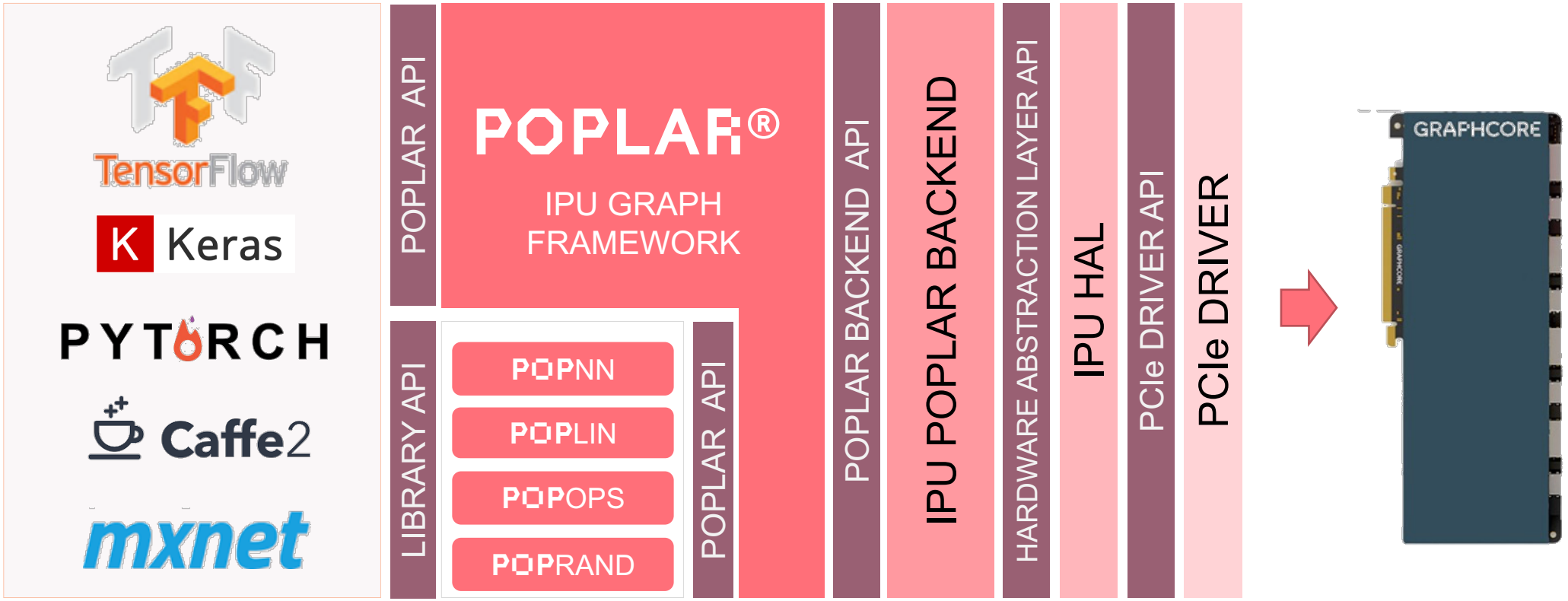


PROGRAMMING THE IPU
POPLAR®

FAMILIAR PROGRAMMING LANGUAGES

High-level graph APIs built on Poplar® native graph abstraction



GETTING STARTED WITH TENSORFLOW

To get started:

Install the Graphcore drivers

Install the Poplar development tools

Install Tensorflow with IPU support

You can also use Keras with a Tensorflow backend

```
$ tar -xzf poplar-1.0.23.tar.gz
$ tar -xzf gc-tensorflow-1.2.34.tar.gz
$ source poplar-1.0.23/enable
(poplar-1.0.23) $ virtualenv ve
(poplar-1.0.23) $ source ve/bin/activate
(poplar-1.0.23) (ve) $ pip install gc-tensorflow-1.2.34/tensorflow.whl
…
```

GETTING STARTED WITH TENSORFLOW

Using Tensorflow is straightforward

No changes to your code are needed

```
import numpy as np
import tensorflow as tf
import imagenet_categories

def run_inference_on_image(model_def, image):
  if not tf.gfile.Exists(image):
    tf.logging.fatal('File does not exist %s', image)

  jpeg_file = tf.gfile.FastGFile(image, 'rb').read()
  tf.reset_default_graph()
  jpeg_input = tf.placeholder(tf.string)
  raw_img = tf.image.decode_jpeg(jpeg_input, channels=3)
  image_data = tf.cast(tf.reshape(raw_img, [1, 224, 224, 3]), tf.float32) / 256.0

  with tf.gfile.FastGFile(model_def, 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    tf.import_graph_def(graph_def, input_map={"input": image_data}, name='')

  with tf.Session() as sess:
    output_tensor = sess.graph.get_tensor_by_name('MnetV1/Predictions/Reshape_1:0')
    predictions = sess.run(output_tensor, {jpeg_input: jpeg_file})
    predictions = np.squeeze(predictions)
    top_k = predictions.argsort()[-5:][::-1]

    for v in top_k:
      print "Class " + str(v-1) + " (" + imagenet_categories.categories[v-1] +
            ") score " + str(predictions[v])

if len(sys.argv) != 3:
  sys.exit("classify_image.py <model_def_file> <image_file>")

files = glob.glob(sys.argv[2]+"*.jpg")
for f in files:
  print "Processing " + f
  run_inference_on_image(sys.argv[1], f)
```

GETTING STARTED WITH TENSORFLOW

Using Tensorflow is straightforward

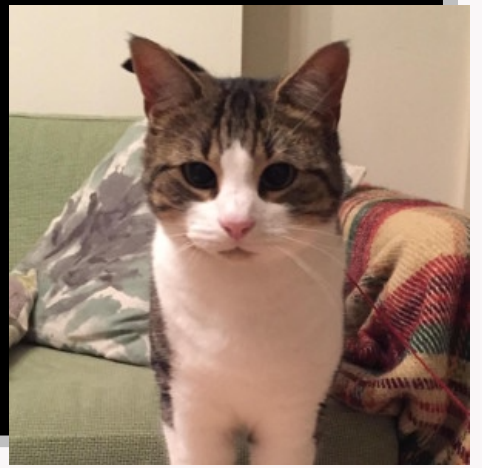
Just execute your python program as normal

```
(ve) $ python classify_image.py --model_dir=model --image_file zeus.jpeg
.
.
.
```

GETTING STARTED WITH TENSORFLOW

When TensorFlow executes its compute graph, the graph gets compiled for the IPU (via the Tensorflow XLA) and then executes

```
(ve) $ python classify_image.py --model_dir=model --image_file zeus.jpeg
 .
 .
 2017-07-03 16:36:29.519189: I poplar/driver/compiler.cc:224] Begin
 compilation of module cluster_40[]_module
 2017-07-03 16:36:29.524397: I poplar/driver/compiler.cc:266] Compiling
 sub-computation %convolution
 2017-07-03 16:36:30.881974: I poplar/driver/compiler.cc:275] Compiling
 main computation cluster_40[].v9
 2017-07-03 16:36:35.181859: I poplar/driver/compiler.cc:295] Compile
 engine cluster_40[]_module
 2017-07-03 16:36:36.316223: I poplar/driver/executable.cc:49] Execute
 cluster_40[]_module
 .
 .
```



GETTING STARTED WITH TENSORFLOW

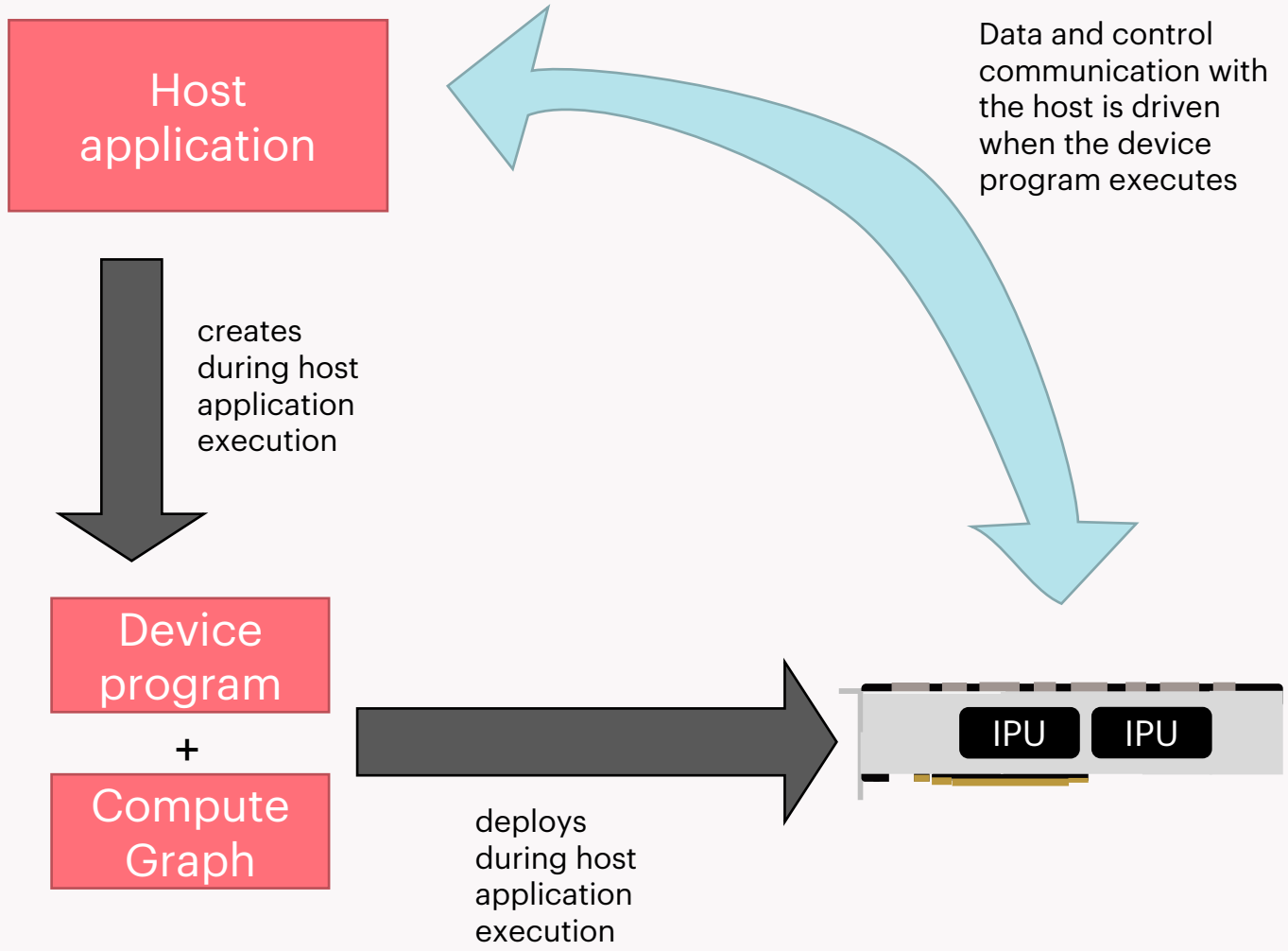
The results are passed from the IPU to the host program

```
(ve) $ python classify_image.py --model_dir=model --image_file zeus.jpeg
.
.
.
.
tabby, tabby cat (score = 0.61839)
Egyptian cat (score = 0.13952)
quilt, comforter, comfort, puff (score = 0.04772)
tiger cat (score = 0.04651)
sleeping bag (score = 0.01473)
```

GETTING STARTED WITH POPLAR®

Poplar is the low level programming library for IPUs

Applications and frameworks can use Poplar to create programs to run on the device with associated compute graphs

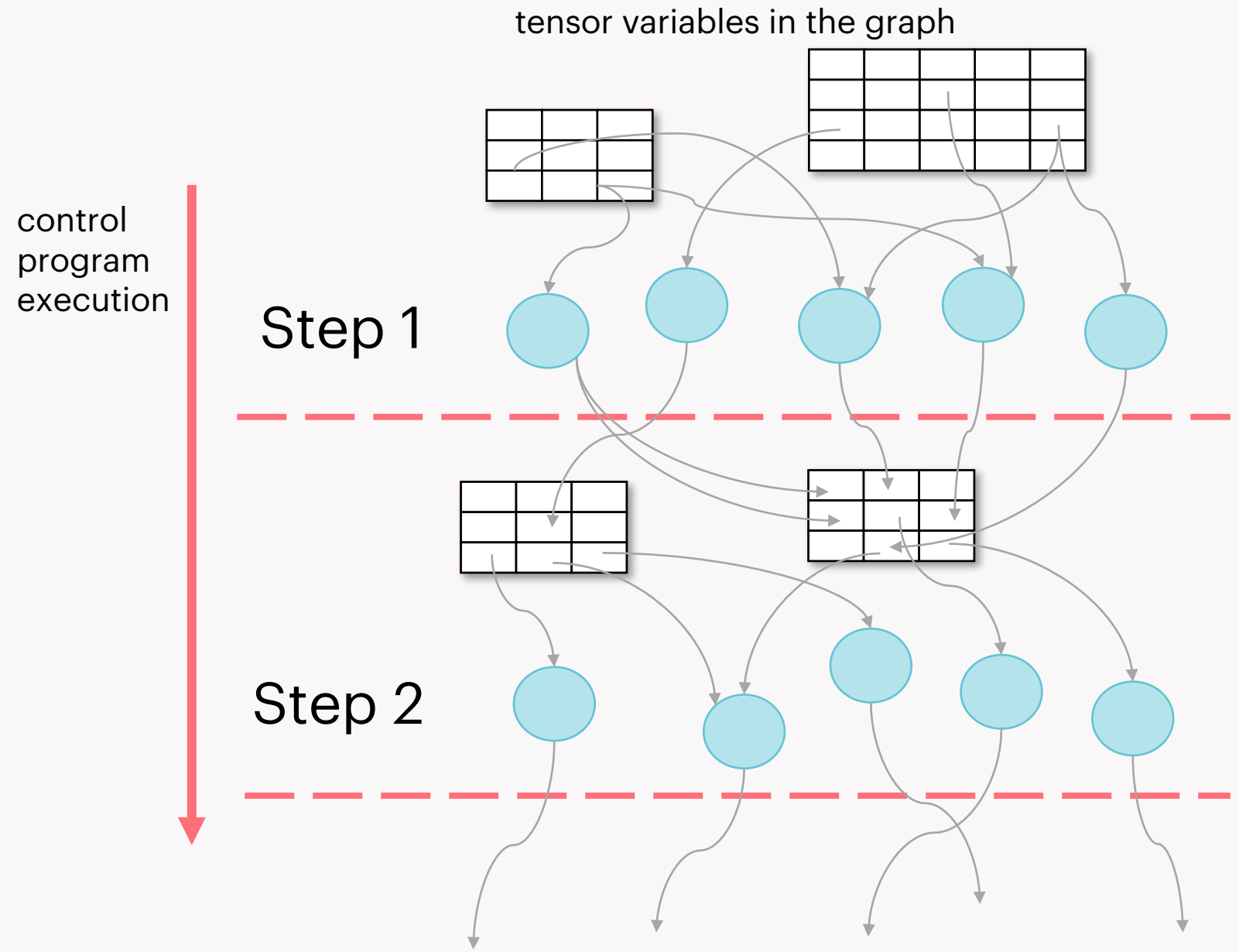


GETTING STARTED WITH POPLAR®

A program is just a sequence of steps to run.

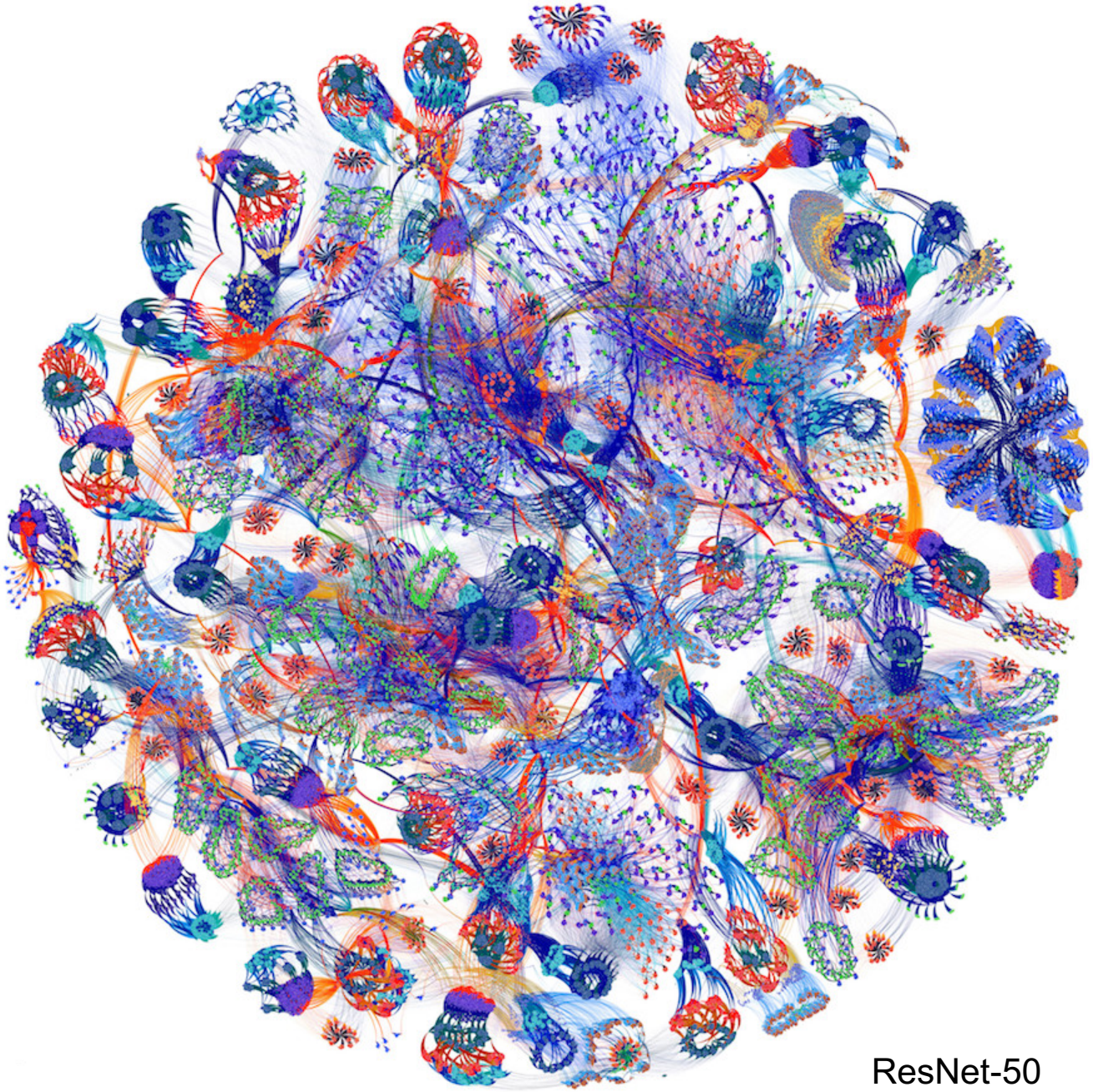
Each step executes many pieces of work in parallel (vertices).

The compute graph describes these vertices along with their data inputs & outputs (edges)



Poplar® graphs have many more vertices than TensorFlow graphs – typically millions, to load-balance a machine executing tens of thousands of codelets in parallel.

The TensorFlow IPU backend uses the Poplar® libraries to break TensorFlow compute functions and large tensors into fragments.



ResNet-50
training batch=4

GETTING STARTED WITH POPLAR®

Poplar is a C++11 library that lets you build up programs and graphs for the device and the run them.

```
#include <poplar/Engine.hpp>

int main() {
// Use the first IPU device on this host
Device device = getDeviceSet().getDevice(0);

// Create a graph targeting the IPU device
Graph graph(device.getTarget());

// Load vertex ‘codelets’ from file
graph.addCodelets(“my-codelets.cpp”);

// Call a user function to define the variables, compute sets and
// vertices in the graph and construct a set of control programs to
// run on the device.
auto progs = constructMyGraph(&graph);

// Construct engine with the constructed control programs and
// graph.
Engine eng(graph, progs);

// Run program 0 on the device via the engine
eng.run(0);
}
```

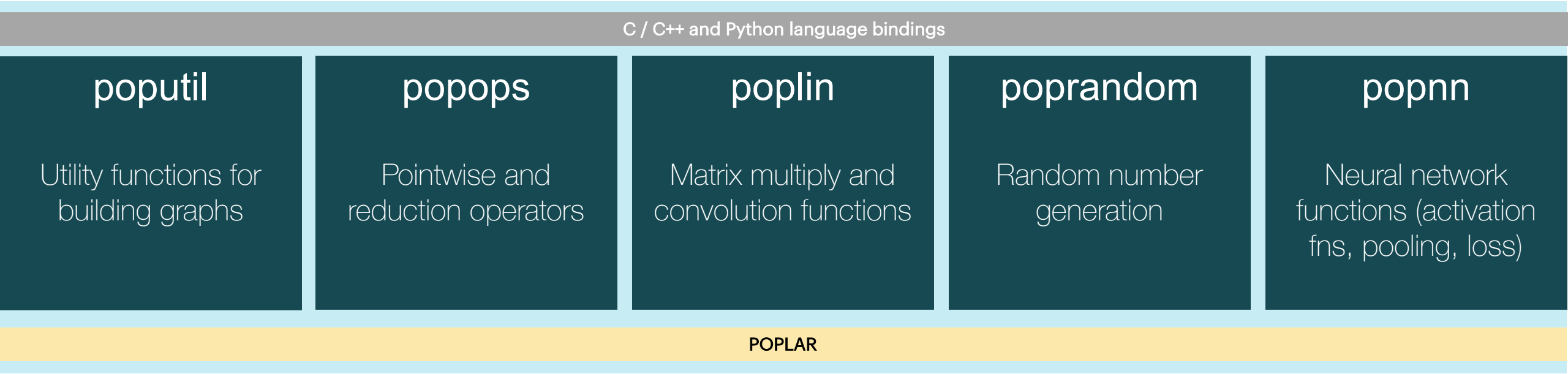
GETTING STARTED WITH POPLAR®

Building an application that uses Poplar is just a matter of compiling a standard C++ application and linking in the Poplar library.

```
$ gcc -std=c++11 my-poplar-program.cpp -lpoplar
$
```

POPLIBS

open source libraries providing functions to add common machine learning operators to your device program and graph IPU devices



GitHub
github.com/graphcore/poplibs

THANK YOU

info@graphcore.ai