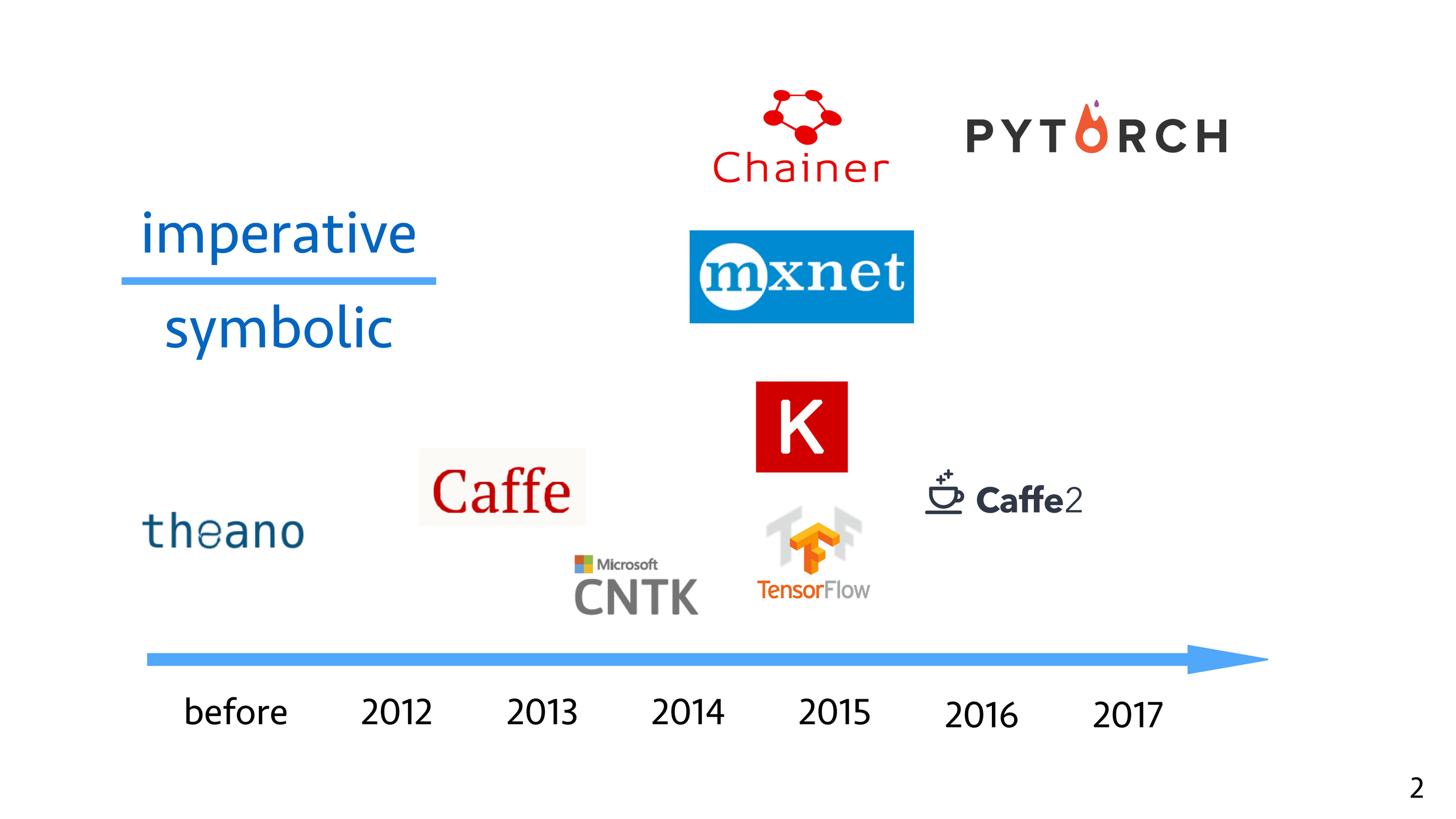
Gluon: new MXNet interface to accelerate research

Mu Li

AWS Deep Learning

https://mli.github.io/cvpr17/


Chainer
PYTORCH
imperative
mxnet
symbolic
K
Caffe
Caffe2
theano
Microsoft
CNTK
TensorFlow
before
2012
2013
2014
2015
2016
2017

Caffe

ResNet-101-deploy.prototxt

```
layer {
        bottom: "data"
        top: "conv1"
        name: "conv1"
        type: "Convolution"
        convolution_param {
                num_output: 64
                kernel_size: 7
                pad: 3
                stride: 2
```

....

(4K lines of codes)

- Protobuf as the interface
- Portable
  - caffe binary + protobuf model
- Reading and writing protobuf are not straightforward

Tensorflow

Implement Adam

```
# m_t = beta1 * m + (1 - beta1) * g_t
m = self.get_slot(var, "m")
m_scaled_g_values = grad.values * (1 - beta1_t)
m_t = state_ops.assign(m, m * beta1_t,
                       use_locking=self._use_locking)
m_t = state_ops.scatter_add(m_t, grad.indices, m_scaled_g_values,
                            use_locking=self._use_locking)
```

> 300 lines of codes

- A rich set of operators (~2000)
- The codes are not very easy to read, e.g. not python-like

Keras

```
model = Sequential()
model.add(Dense(512, activation='relu',
                input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.compile(...)
model.fit(...)
```

- Simple and easy to use
- Difficult to implement sophisticated algorithms

Pytorch & Chainer

```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

- Flexible
- Complicate programs might be slow to run

MXNet

Implement Resnet

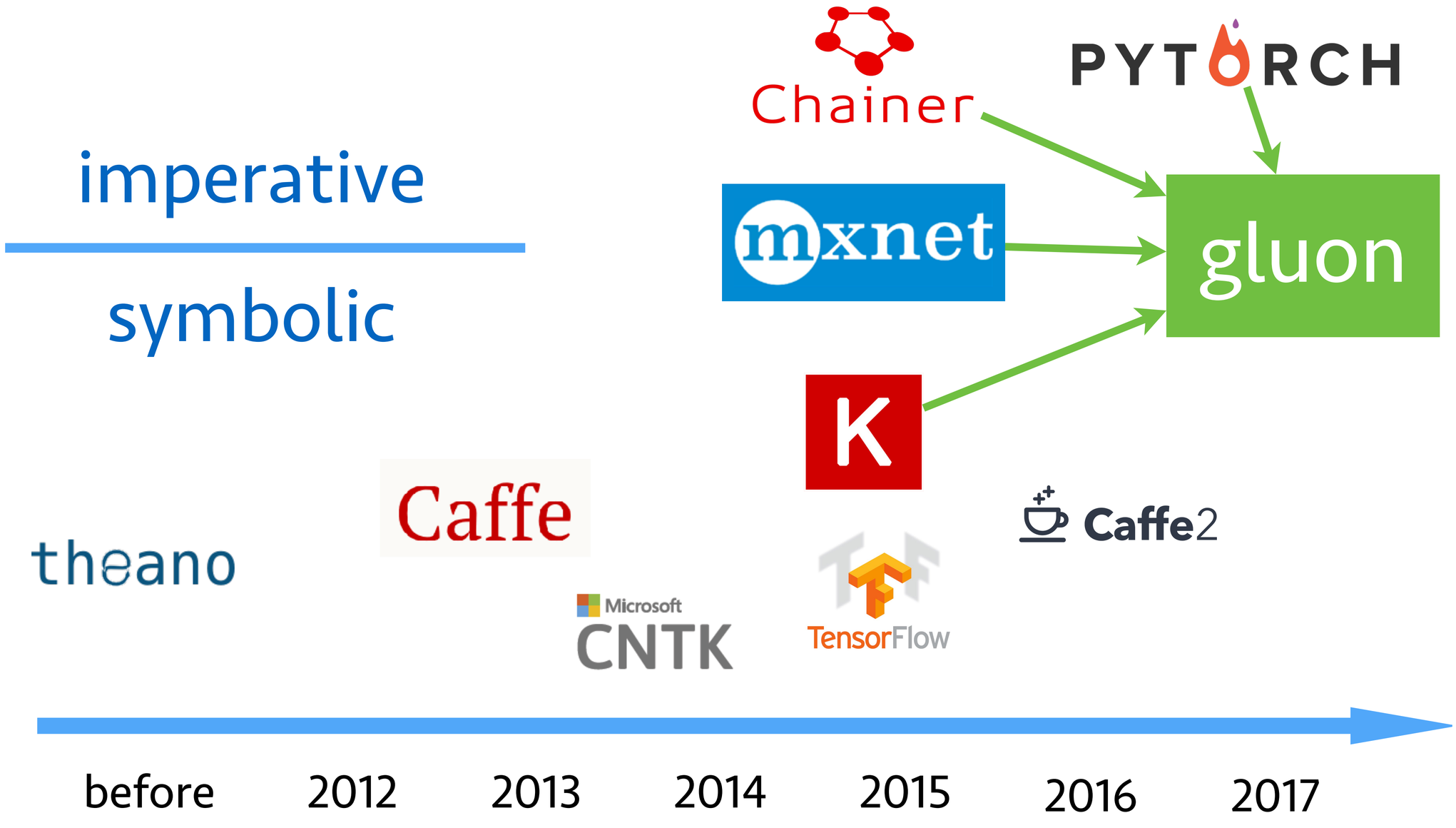
```
bn1 = sym.BatchNorm(data=data, fix_gamma=Fal
act1 = sym.Activation(data=bn1, act_type='re
conv1 = sym.Convolution(data=act1, num_filte
```

Implement Adam

```
coef2 = 1. - self.beta2**t
lr *= math.sqrt(coef2)/coef1

weight -= lr*mean/(sqrt(variance) + self.epsilon)
```

- Symbolic on network definition
- Imperative on tensor computation
- Huh.., not good enough


imperative
symbolic
Chainer
PYTORCH
mxnet
gluon
K
Caffe
theano
Caffe2
Microsoft
CNTK
TensorFlow
before
2012
2013
2014
2015
2016
2017

Gluon at a glance

```
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Dense(128, activation='relu'))
    net.add(gluon.nn.Dense(64, activation='relu'))
    net.add(gluon.nn.Dense(10))

loss = gluon.loss.SoftmaxCrossEntropyLoss()

for data, label in get_batch():
    with autograd.record():
        l = loss(net(data), label)
    l.backward()
    trainer.step(batch_size=data.shape[0])
```

`net.hybridize()` converts from imperative to symbolic execution

In summary

- Symbolic
  - efficient & portable
  - but hard to use
- Gluon
  - imperative for developing
  - symbolic for deploying
- Imperative
  - flexible
  - may be slow