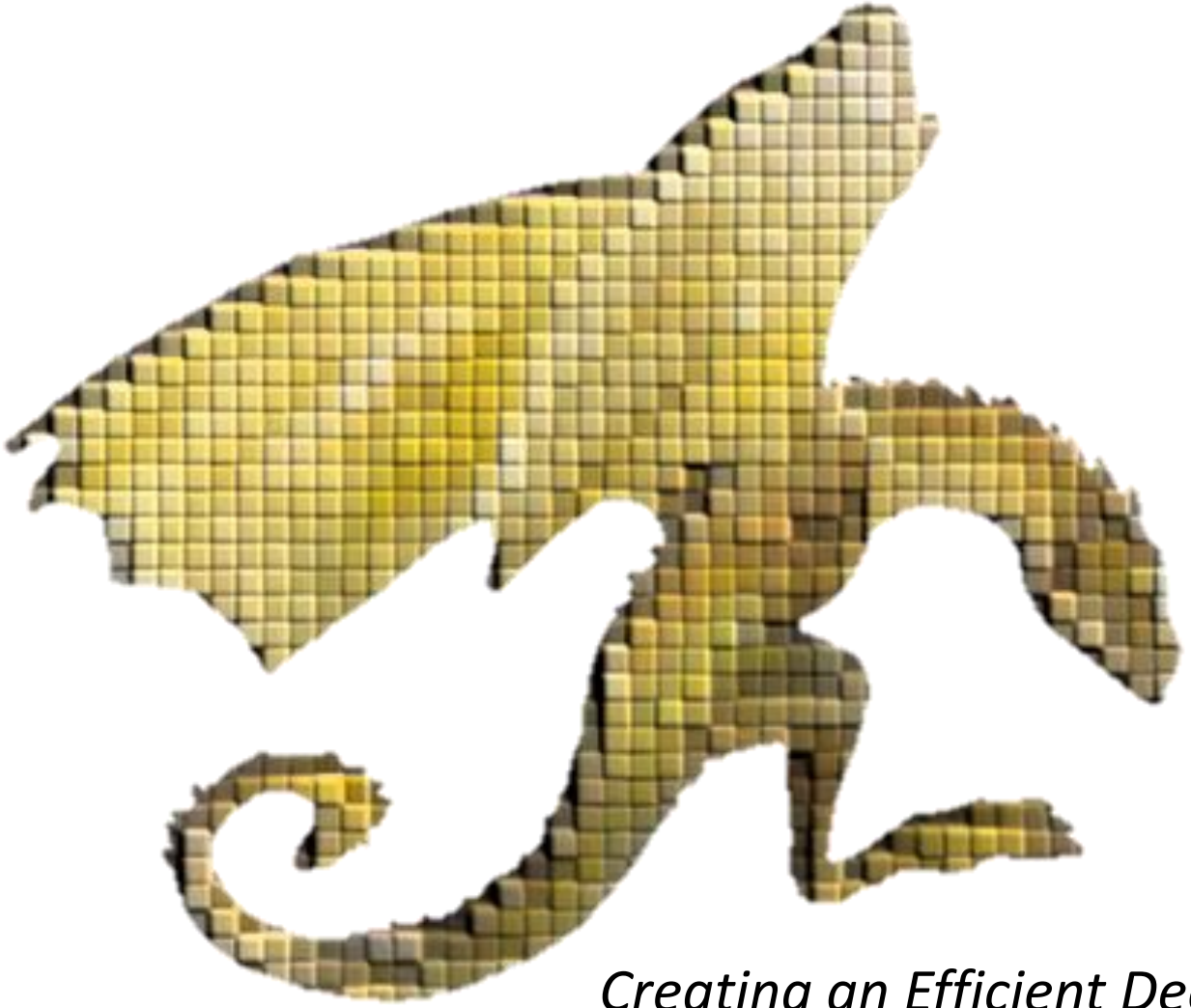


Viserion

Creating an Efficient Deep Learning Pipeline for Prism in torch7

Why torch & lua

- Used extensively in industry (facebook, google, twitter, etc. …)
- Extremely fast (Close to C), easy to extend
- Can call ANY C function / library
- Many existing Libraries
  - Cephes
  - Signal Processing
  - Autograd
  - Reinforcement Learning
  - SVM, PCA, etc..
  - Caffe
  - OpenCV
  - …

Quick Note: Lua Tables

Tables are associative arrays ➔ {key, value} pairs

```
myTable = {}
myTable[1] = “Hello World”

myTable[“foo”] = “Hello Prism”
```

SAME AS (for strings)

```
myTable.foo = “Hello Prism”
```

Goals

- Code Comprehension
- (Re)Usability
- Efficiency
- Information Sharing
- Focus on ML, not software engineering!

Design Principles

- Lightweight
- No restrictions on Lua libraries
- No ‘special’ syntax

➔ Everything you find online is usable!

Viserion is:

- A collection of lua / c / c++ libraries
- A framework to organise deep learning code

It takes care of

- Multithreading
- CPU/GPU data transfers
- Multi-GPU execution
- Easier debugging

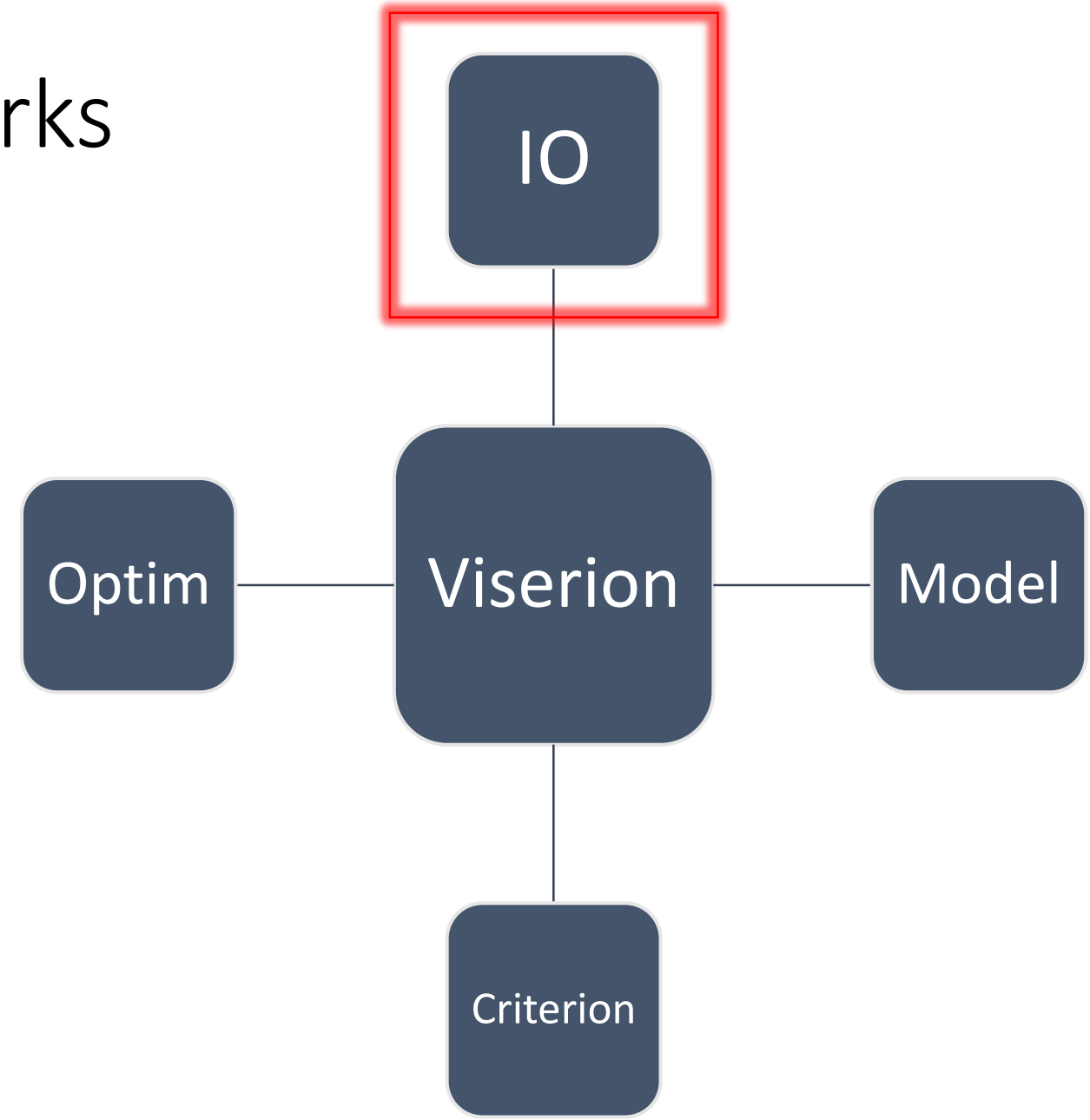
How it works

- Provide 4 scripts for:
  - IO
  - ‘Model’ ➔ actual architecture of your model
  - ‘Criterion’ ➔ loss function
  - Optimiser

- Call Viserion.lua & we take care of the rest!

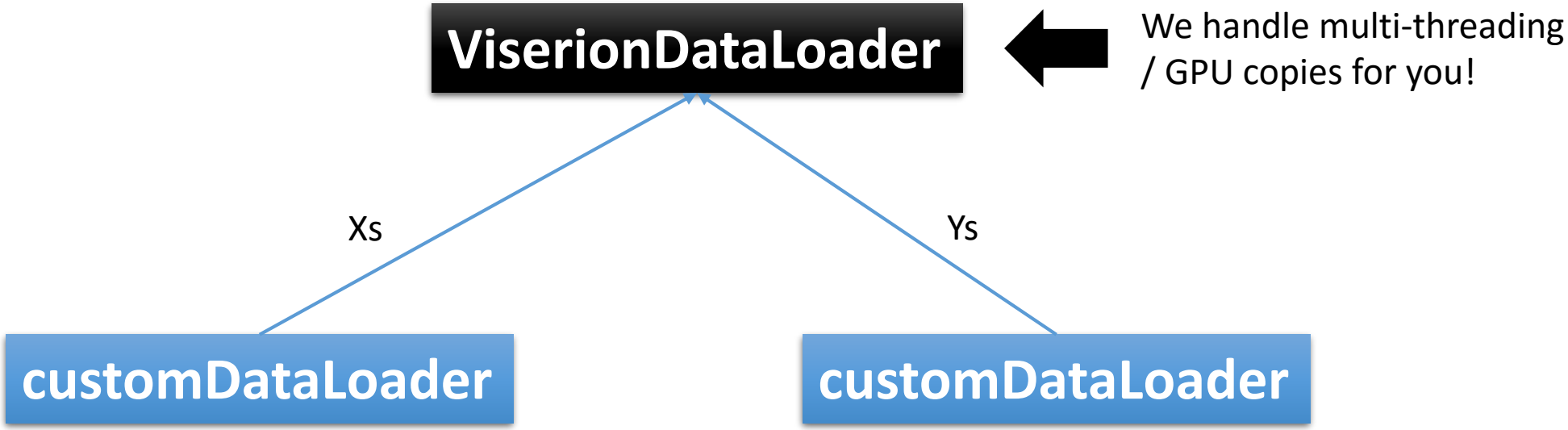
- **<u>Note:</u>** *All in global namespace, so you have access to everything (model, optimiser, IO) across scripts/inside functions!*

How it works



IO - Overview

We need to define how to get pairs {X, Y} for training/validation:



IO – Lua script

```lua
local superDataLoader = require 'Viserion/ViserionDataLoader'
local ViserionMNISTLoader = require 'Viserion/dataLoaders/ViserionMNISTLoader'

trainXPath = 'data/train-images.idx3-ubyte'
trainYPath = 'data/train-labels.idx1-ubyte'

testXPath = 'data/t10k-images.idx3-ubyte'
testYPath = 'data/t10k-labels.idx1-ubyte'

trainXLoader = ViserionMNISTLoader(trainXPath, false)
testXLoader = ViserionMNISTLoader(testXPath, false)

trainYLoader = ViserionMNISTLoader(trainYPath, true)
testYLoader = ViserionMNISTLoader(testYPath, true)


trainDataLoader = superDataLoader(opts, trainXLoader, trainYLoader)
testDataLoader = superDataLoader(opts, testXLoader, testYLoader)

function saveState(epoch, loss, testOutput)
        print('Save State Not Implemented...')
end
```

1

Your dataloaders

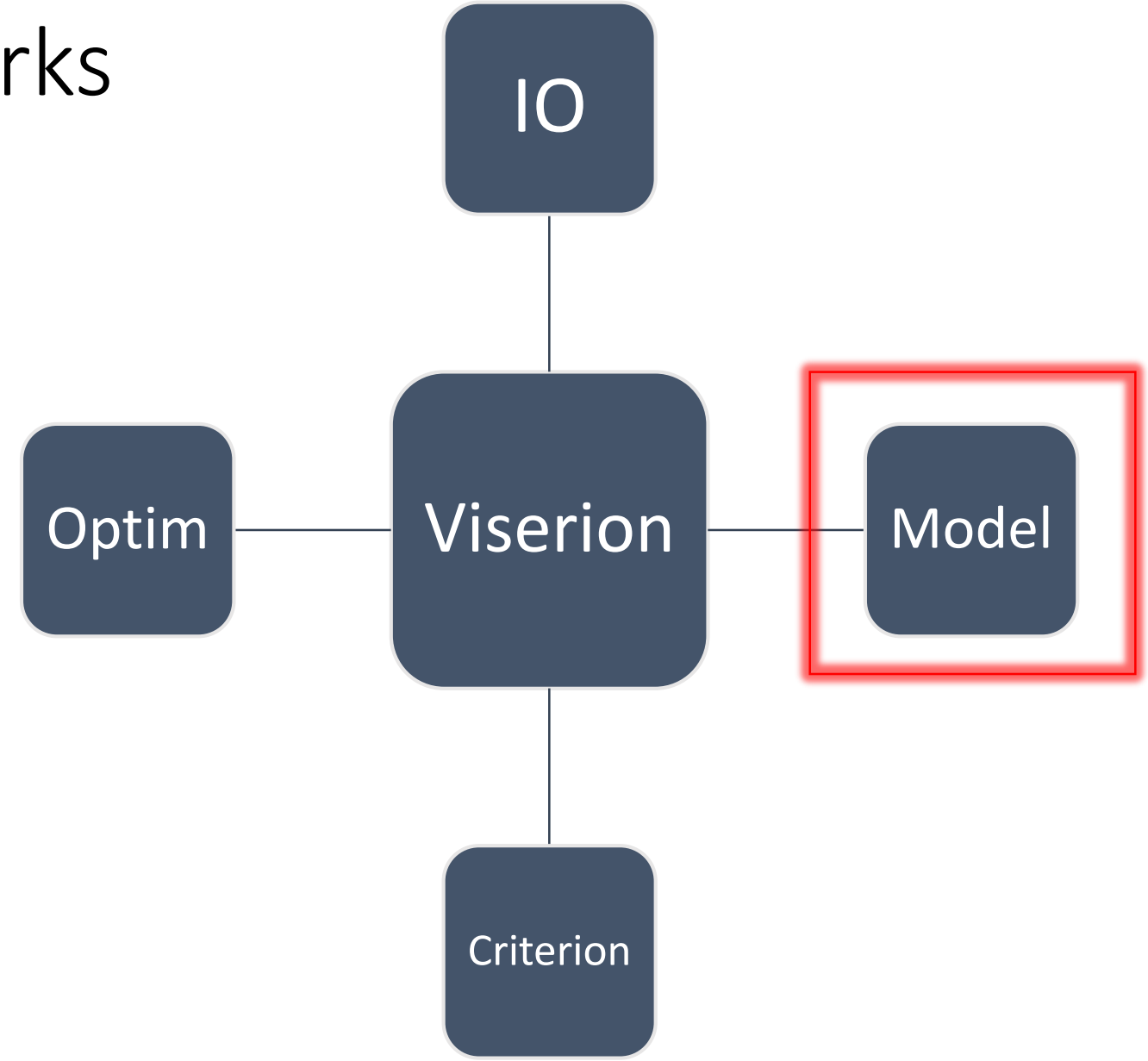
2

Just pass them as arguments to the ViserionDataLoader!

3

Gets called when (and if) you want

How it works

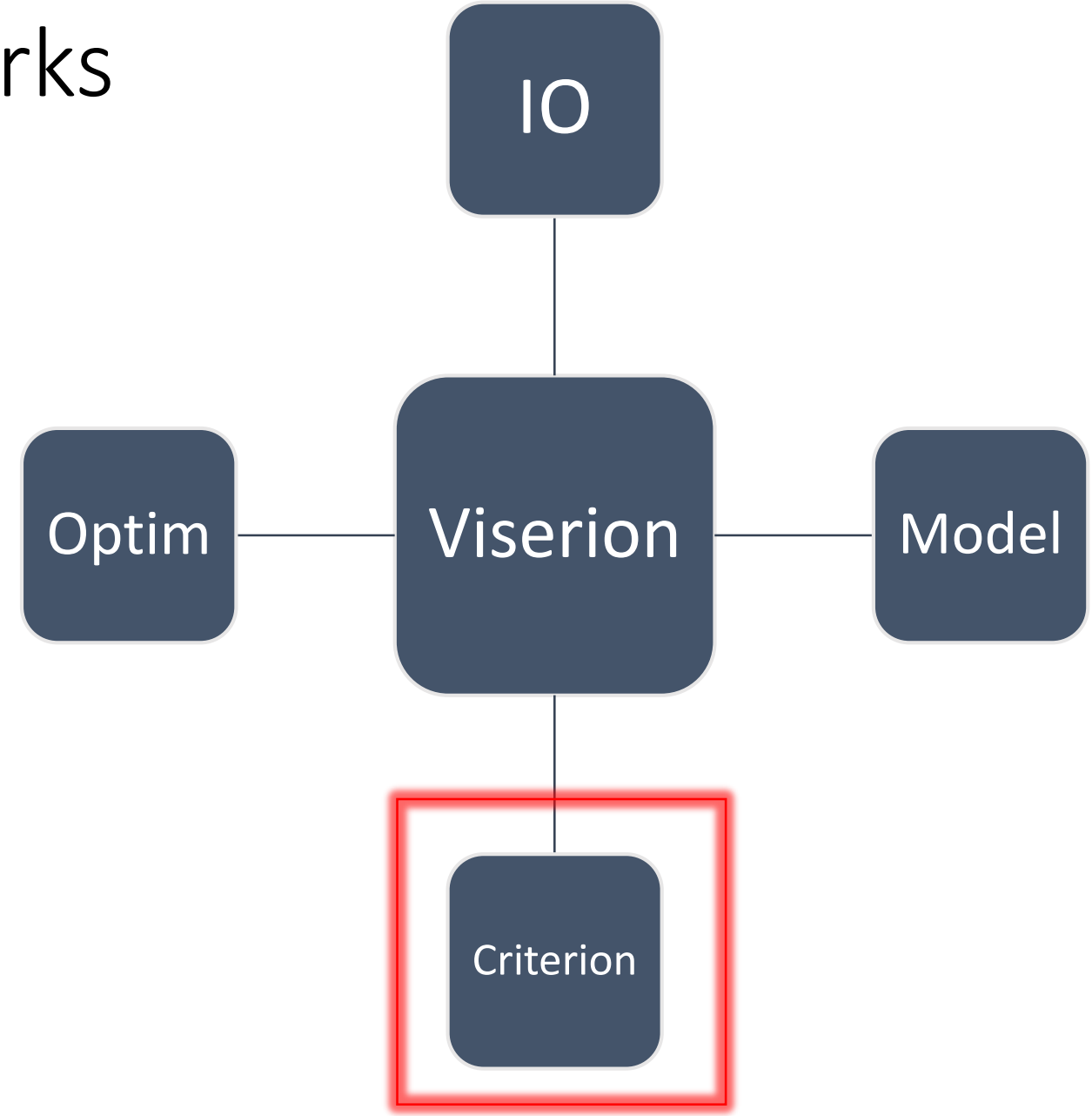


Model – Lua Script

```
1   require 'nn'
2
3 1 model = nn.Sequential()
4
5   model:add(nn.SpatialConvolutionMM(1, 32, 5, 5))
6   model:add(nn.Tanh())
7   model:add(nn.SpatialMaxPooling(3, 3, 3, 3))
8   -- stage 2 : mean suppresion -> filter bank -> squashing -> max pooling
9   model:add(nn.SpatialConvolutionMM(32, 64, 5, 5))
10  model:add(nn.Tanh())
11  model:add(nn.SpatialMaxPooling(2, 2, 2, 2))
12  -- stage 3 : standard 2-layer MLP:
13  model:add(nn.Reshape(64*2*2))
14  model:add(nn.Linear(64*2*2, 200))
15  model:add(nn.Tanh())
16  model:add(nn.Linear(200, 10))
17
18  -- from https://github.com/torch/demos
```

Your model

How it works

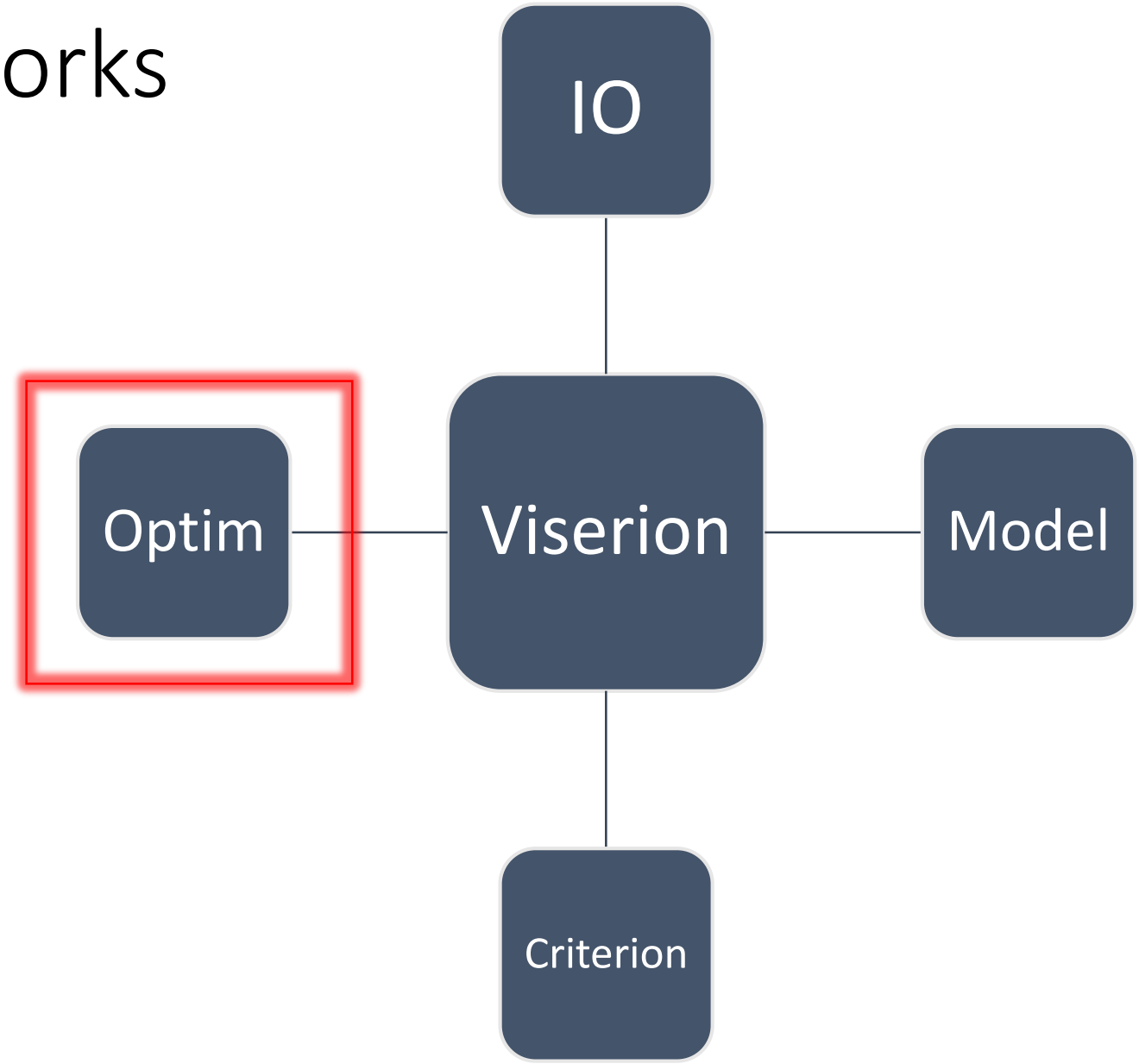


Criterion – Lua Script

```
1   require 'nn'
2
3 1 criterion = criterion = nn.DistKLDivCriterion()
```

Your criterion

How it works



Optimisation – Lua Script

```
1   require 'optim'
2
3 1 optimOptimiser = optim.nag
4
5 2 optimConfig = {}
6
7   optimConfig.learningRate = 1e-3
8   optimConfig.learningRateDecay = 0.0
9   optimConfig.weightDecay = 0.0
10  optimConfig.momentum = 0.98
11  optimConfig.dampening = 0.0
12
13 3 function defineCustomLearningRate(epoch, currentLearningRate)
14      return currentLearningRate
15  end
```

Which optimiser?

Which options for that optimiser?

Controlling the LR

Debugging

Use *-debug*

➔Prints sizes, types, helpful hints

Include calls to `printDebug(<>)` anywhere in your code

➔We will *only* print this when *'–debug'* is used

<u>Note also:</u> `printDebug(), printWarning(), printError()`

Custom DataLoaders

- Any lua class with the following functions:

```
function ViserionMNISTLoader:size()
    --...
end

function ViserionMNISTLoader:getNarrowChunk(dim, index, size, batchNum, epoch)
    --...
end

function ViserionMNISTLoader:getNarrowChunkNonContiguous(dim, idxList, batchNum, epoch)
    --...
end
```

1 **torch.LongStorage size of the WHOLE dataset, as a tensor**

2 **returns tensor /table (pointer to existing)**

3 **returns tensor /table (pointer to existing)**

Use for data-augmentation to seed random numbers across threads using `torch.manualSeed()`

Advanced Mode – Custom Model Flow

You can control

- Order of criteria evaluation (forward + backwards passes)
- Backwards pass over model

➔Define a table `criteria` instead of `criterion`

➔Define 4 functions

Advanced Mode – Custom Model Flow

```
function defineCriteriaFlowForward(idx, modelForward, input, target, previousCriteriaForward)
  if idx == 1 then
    --..
  elseif idx == 2 then
    --..
  end
end

function defineCriteriaFlowBackward(idx, modelForward, input, target, criteriaForward, previousCriteriaBackward)
  if idx == 1 then
    --..
  elseif idx == 2 then
    --..
  end
end

function defineModelFlowBackward(modelForward, criteriaBackward, input, target)
  --..
end

function defineCriteriaPrintOut(epoch, isTraining, criteriaForwards)
  --..
end
```

1

2

3

4

Print to terminal at each epoch

Viserion executes: (sequentially!)

```
function defineCriteriaFlowForward(idx
  if idx == 1 then
    --..
  elseif idx == 2 then
    --..
  end
end

function defineCriteriaFlowBackward(id
  if idx == 1 then
    --..
  elseif idx == 2 then
    --..
  end
end

function defineModelFlowBackward(model
  --..
end
```

1

2

3

```
model:forward()
```

For_each criterion in criteria:

```
criterion:forward(defineCriteriaFlowForward())
```

For_each criterion in criteria:

```
criterion:backward(defineCriteriaFlowBackward())
```

```
model:backward(<Xs>, defineModelFlowBackward())
```

How to run?

- Clone Viserion
- Create your own repo
- Create a symlink to Viserion directory

```
th Viserion/Viserion.lua -ioFile <> -modelFile <>
-criterionFile <> -optimFile <> -batchSize 10
-numThreads 6 -doTraining
```

You have access to cmd args ANYWHERE in the 'opts' table

Other args

- We assume regression by default, use `-printCLerrors` to get error rates as well
- Specify GPUs with `–specifyGPUs 34` (uses 3 AND 4, with 4 as host)
- Use `–customDataLoaderFile` to link your own classes (you can return more than one from a single file)
- Use `-modelName` <> to customise printouts
- And loads more, don’t forget about `-debug`

Currently in Development

In beta (available in 1-2 weeks)

- Faster Image IO for exr/png & data-preprocessing in c++

In alpha (available in 3-4 weeks)

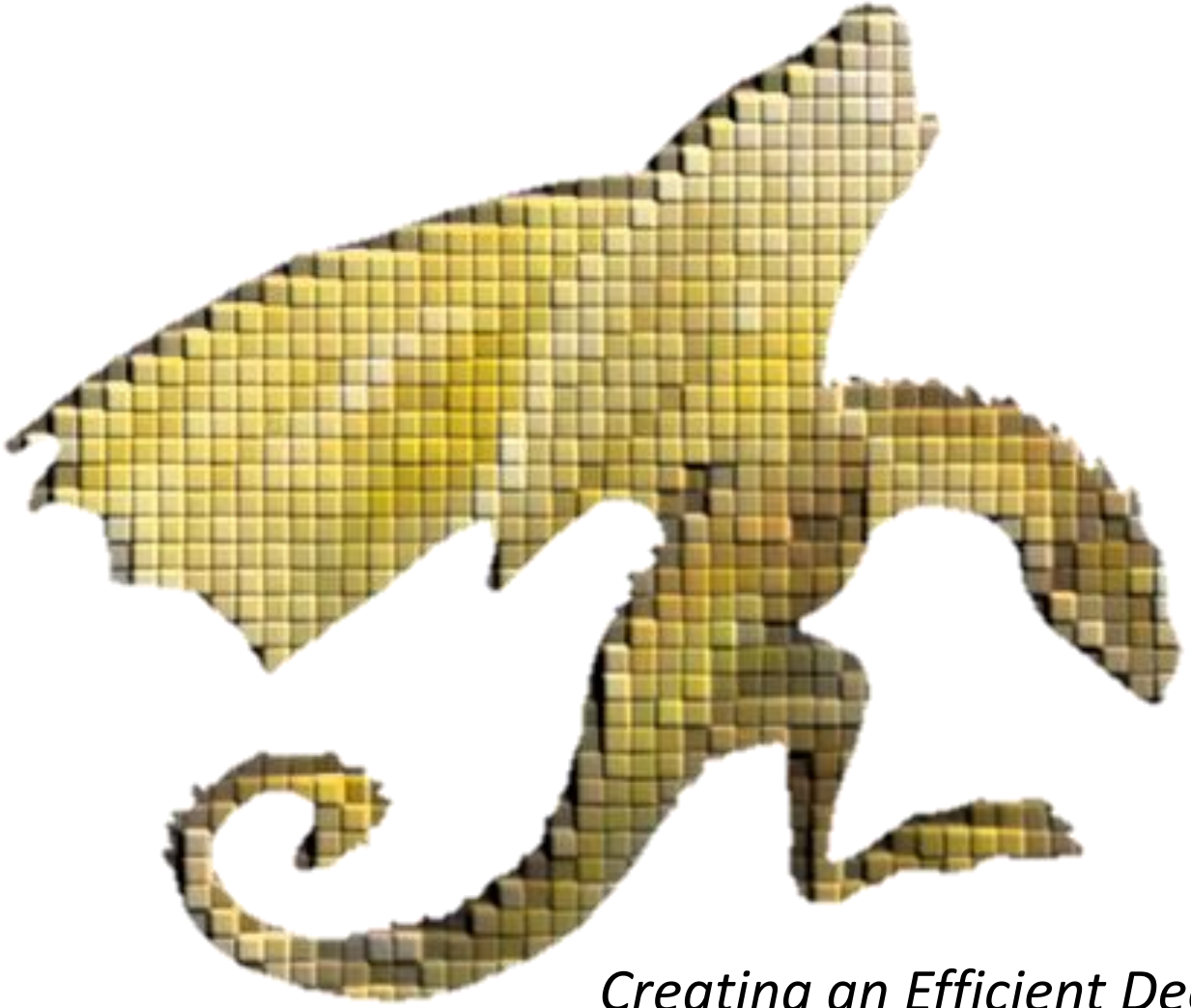
- Viserion in your browser (no more need for ssh), including graphing in javascript, dynamic system monitor, ‘intellisense’ etc.

Future features

- Node editor for neural networks/IO ➔ compilation into Tensorflow

Final Comments

- Report bugs, comments, feature requests, questions on git / slack to me
- Email me if you want to be in the mailing list
- There will be a presentation on each new update/feature
- Let me know if you want access to beta features
- Keep stuff in the group until I release it fully to the public



Viserion

Creating an Efficient Deep Learning Pipeline for Prism in torch7