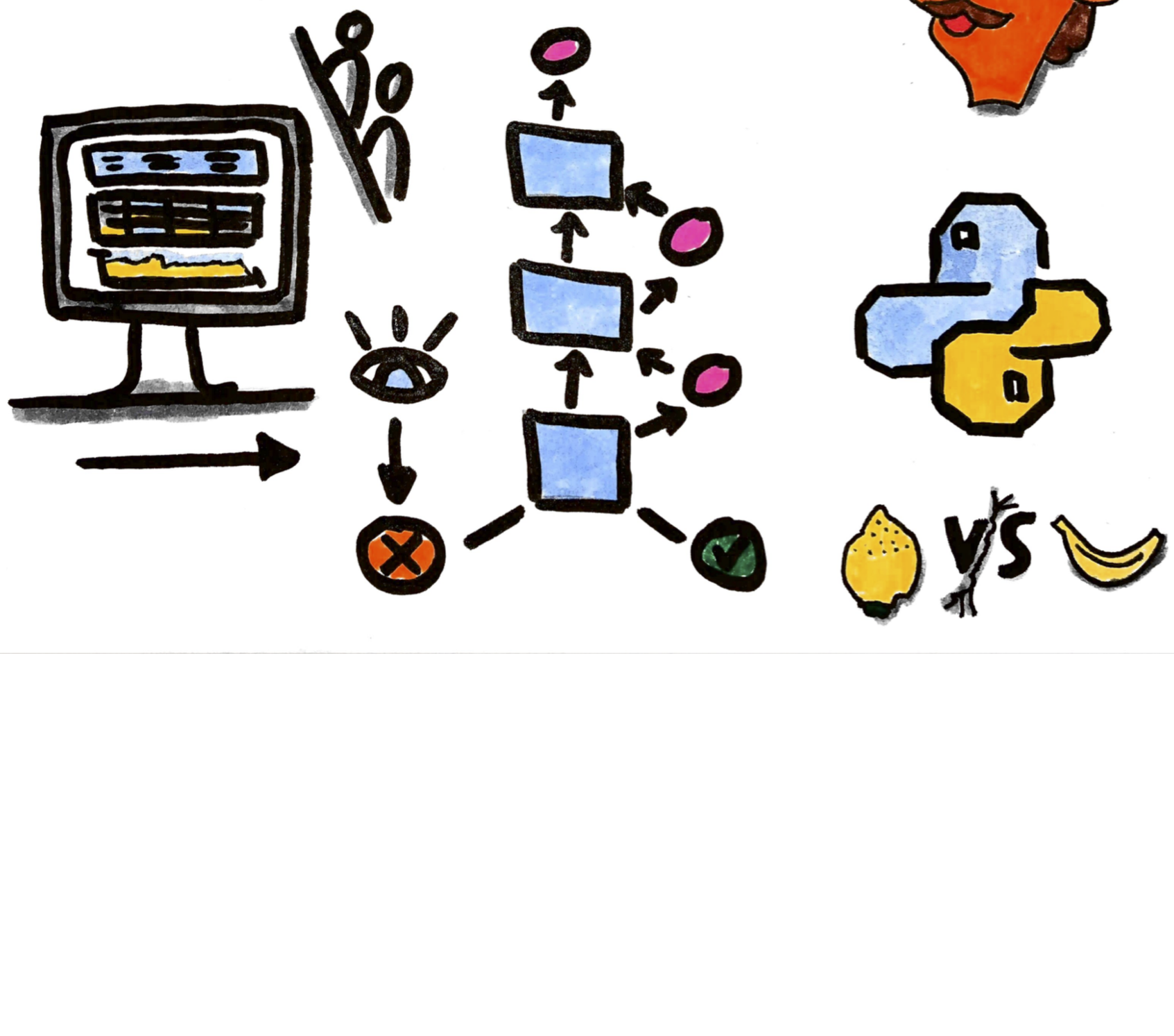
Production ready Data Science



About this Workshop

Interactively build a model that differentiates between limes and bananas
Get a glimpse of production-readiness
Learn about Luigi pipelines and their main components
Write your production ready pipeline
Get an overview of luigis modules
Take a glimpse at TensorFlow Serving for model deployment

Building the model interactively

Download dataset

In [ ]:
```
!mkdir -p dataset
!curl -L -o dataset/dataset_v1.zip http://plainpixels.work/resources/datasets/data
set_v1.zip
!unzip -u -q -d dataset/fruit-images-dataset dataset/dataset_v1.zip
```

In [7]:
```
!ls dataset/fruit-images-dataset/Test
```

```
Banana Lemon
```

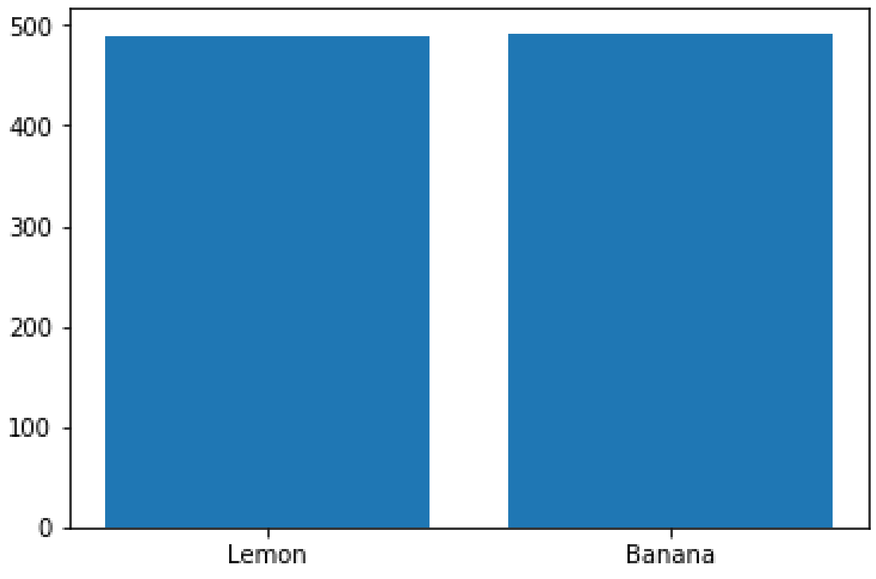
Explore the data

In [3]:
```
img_height, img_width = 100, 100

images_train_path = 'dataset/fruit-images-dataset/Training/'
images_test_path = 'dataset/fruit-images-dataset/Test/'

imageGenTrain = ImageDataGenerator(rescale=1. / 255)
imageGenTest = ImageDataGenerator(rescale=1. / 255)
train_data = imageGenTrain.flow_from_directory(images_train_path,
                                               target_size=(img_height, img_width
), color_mode='rgb', batch_size=32)
test_data = imageGenTest.flow_from_directory(images_test_path,
                                             target_size=(img_height, img_width),
color_mode='rgb', batch_size=32)
```

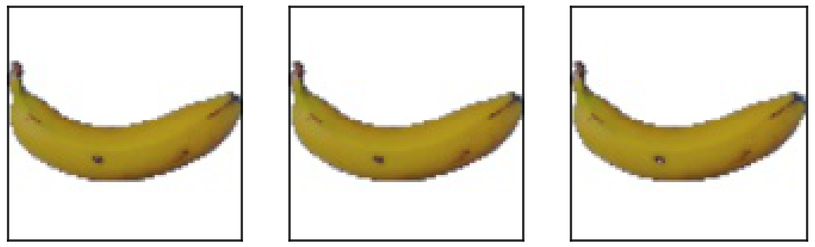
```
Found 982 images belonging to 2 classes.
Found 330 images belonging to 2 classes.
```

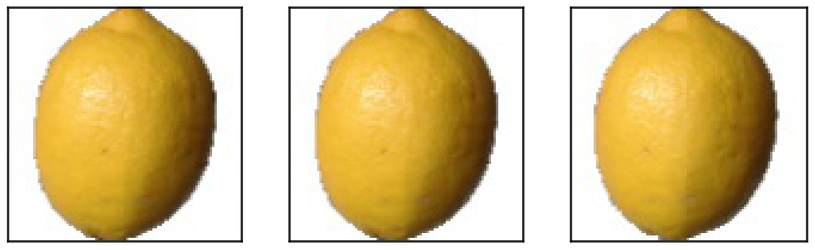
In [4]:
```
input_shape = train_data.image_shape
classes = train_data.class_indices
num_classes = len(classes)
class_counts = numpy.unique(train_data.classes, return_counts=True)[1]
print "shape  : \t(%s, %s, %s)" % input_shape
print "classes: \t%s, %s" % (num_classes, classes)
print "size   : \t%s" % train_data.n
chart = plt.bar(classes.keys(), class_counts)
plt.show(chart)
```

```
shape  :        (100, 100, 3)
classes:        2, {'Lemon': 1, 'Banana': 0}
size   :        982
```



In [5]:
```
plot_images(start_from=0, rows=1)
plot_images(start_from=170, rows=1)
```





Build a baseline model

Distinguish Limes from Bananas using OpenCV

In [6]:
```
path = "dataset/fruit-images-dataset/Test/"
correct = 0.0
for i in range(0, test_data.n):
    label = test_data.classes[i]
    img = cv2.imread("%s/%s" % (path, test_data.filenames[i]))
    im = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    circles = cv2.HoughCircles(im, cv2.HOUGH_GRADIENT,
                               dp=2, minDist=15, param1=100, param2=70)
    if circles is not None:
        if label == 1:
            correct += 1
    elif circles is None:
        if label == 0:
            correct += 1

print "Accuracy: %f" % (correct/test_data.n)
```

```
Accuracy: 0.957576
```

Use a deeplearning model

Create the model using Keras

In [19]:
```
model = Sequential()
model.add(Conv2D(filters=2, kernel_size=(3, 3), strides=1, activation='relu', inpu
t_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.5))
model.add(Flatten())
model.add(Dense(units=16, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units=num_classes, activation='softmax'))
model.compile(loss = 'categorical_crossentropy',
              optimizer = keras.optimizers.RMSprop(),
              metrics = ['accuracy'])
```

Train the model

In [20]:
```
epochs = 5
steps_per_epoch = train_data.samples // train_data.batch_size
validation_steps = test_data.samples // test_data.batch_size

history_dict = model.fit_generator(
  train_data,
  steps_per_epoch=steps_per_epoch,
  epochs=epochs,
  verbose=3
)

print(history_dict.history['acc'])
```

```
Epoch 1/5
Epoch 2/5
Epoch 3/5
Epoch 4/5
Epoch 5/5
[0.5957894736842105, 0.7452631580202203, 0.9726315789473684, 0.9895833333333333
4, 0.9884210522551286]
```

Evaluate the model

In [22]:
```
evaluation = model.evaluate_generator(test_data)
print evaluation
```

```
[0.018613455768949776, 0.9939393939393939]
```

It works, now DEPLOY it!


Integration Testing
Validate Quality
Reproducable Training
Reviews
Test Regressions
Unit Testing
Production
CI/CD
Collect Metrics
Alerting
Scheduling
Proper Logging
Monitoring

A bit about Luigi

Luigi helps to stitch long running tasks together into pipelines

It contains a wide toolbox of task templates (e.g. Hive, Pig, Spark, Python)

How to compose workflows?

A workflow consists of Tasks, Targets and Parameters

Targets correspond to a file or a database entry or some other kind of checkpoint

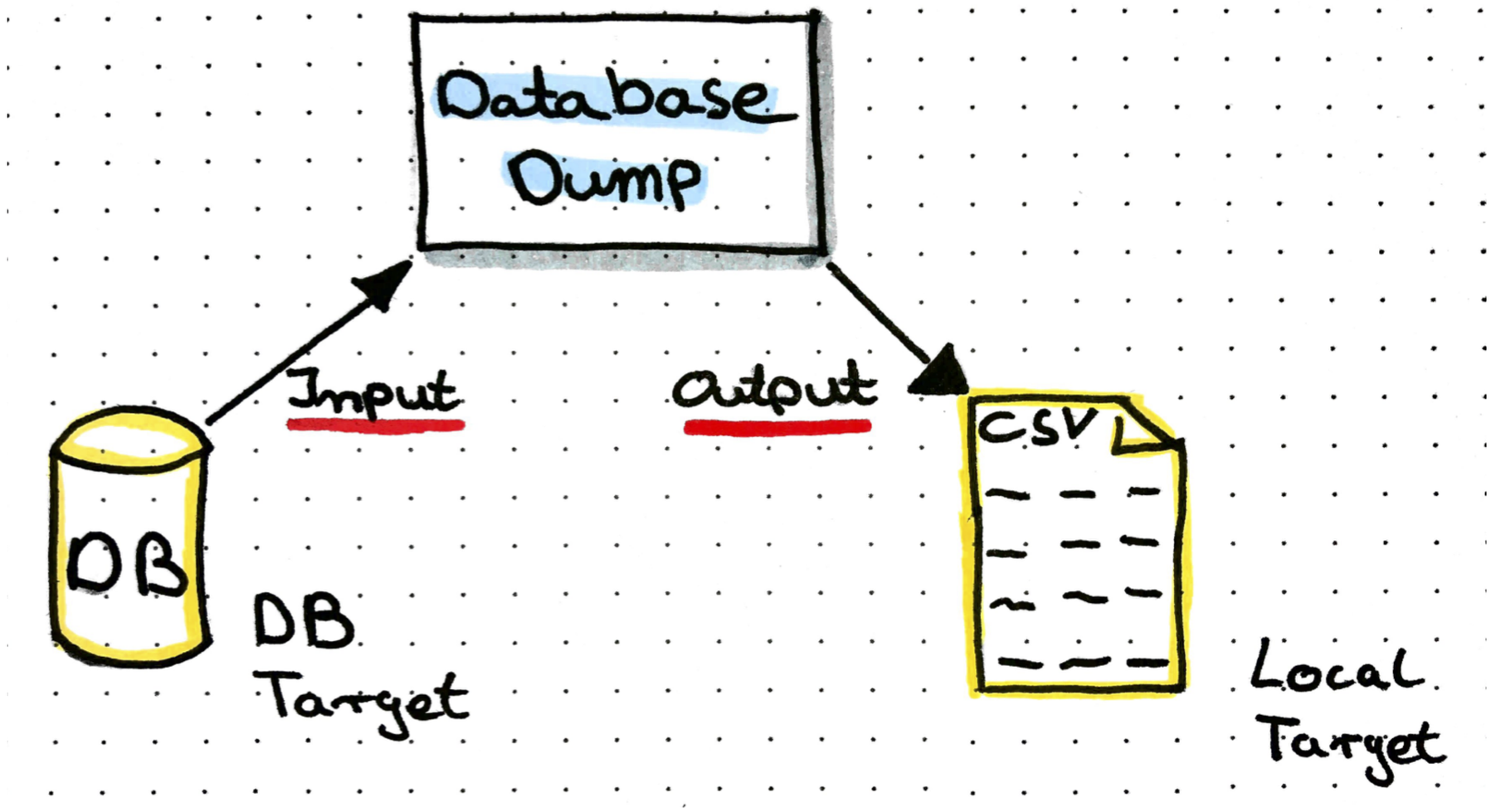
Tasks consume Targets of other tasks, run a computation, then output a target

Parameters take care of task parameterization

Targets

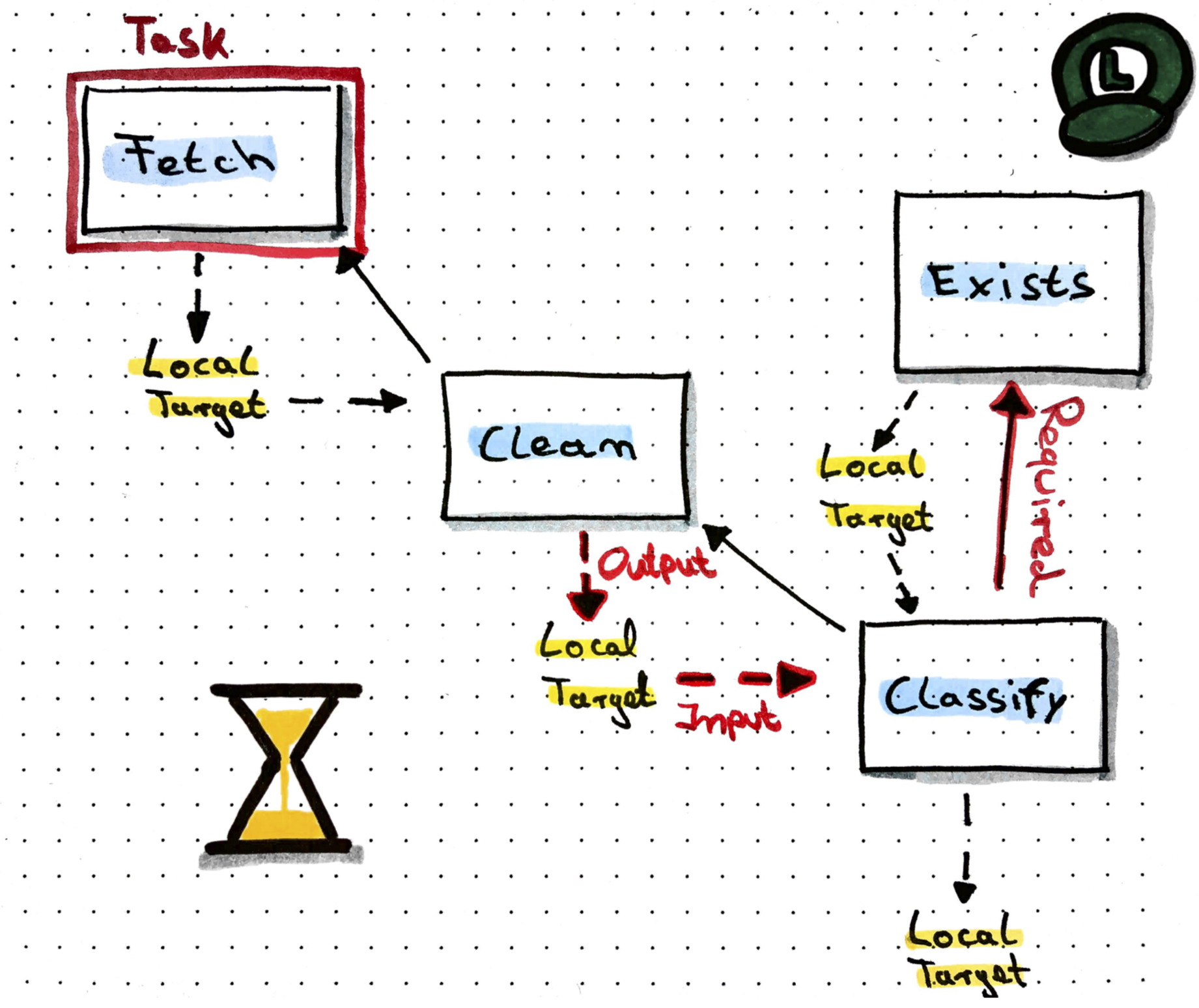
- Files on disk or database entries
- Checkpoints that prevent tasks from multiple executions
- A lot of implementations already exist in the Luigi framework
- LocalTarget (File), RemoteTarget (SSH), HDFSTarget, MySqlTarget, ...

Targets



Tasks

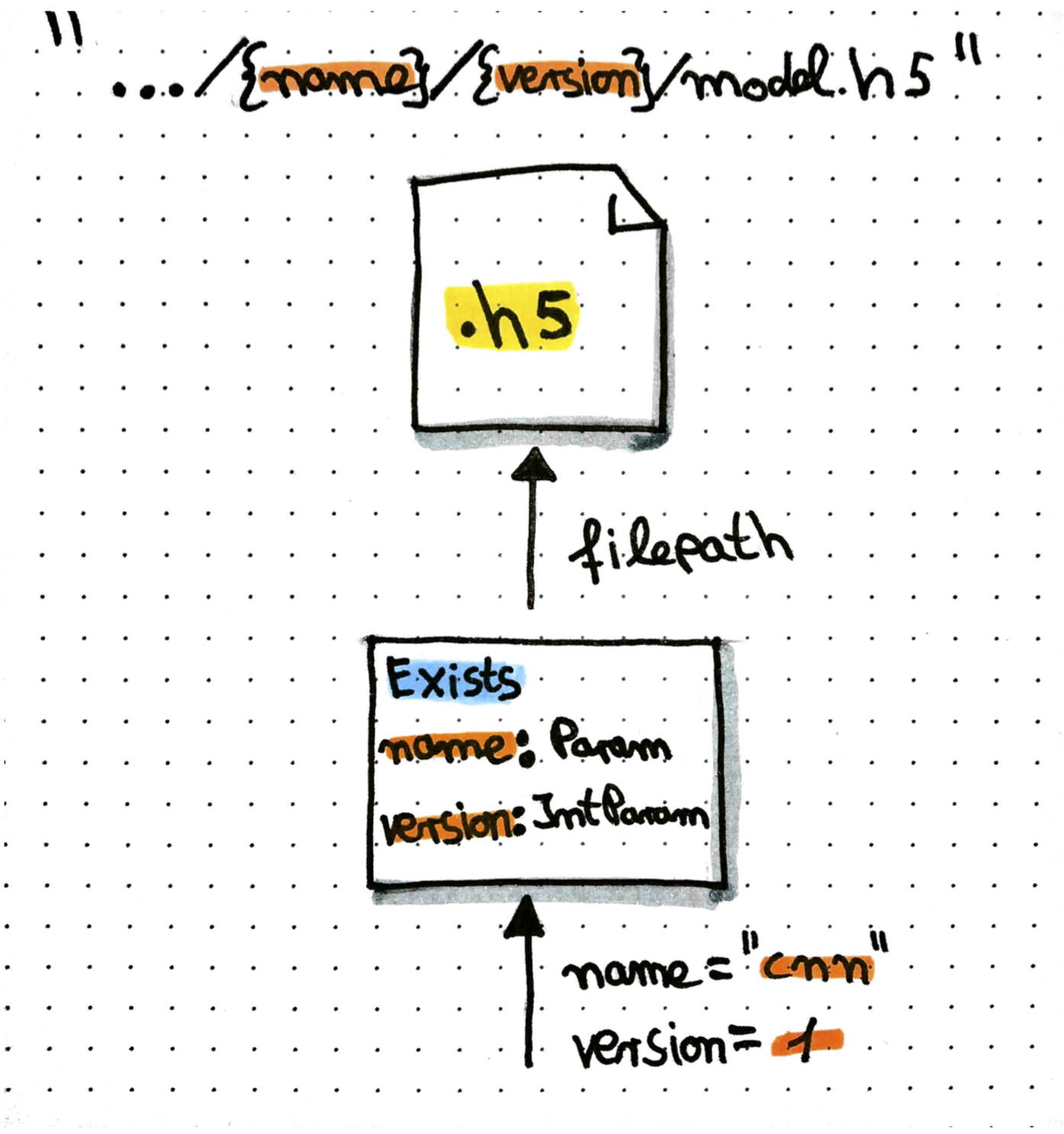
- Implement the actual processing
- Consume targets, process data and save results in new target
- Respect dependencies to other tasks
- Implemented via Python-Classes


Task
Fetch
Local
Target
Clean
Output
Local
Target
Input
Exists
Local
Target
Require
Classify
Local
Target

Parameters

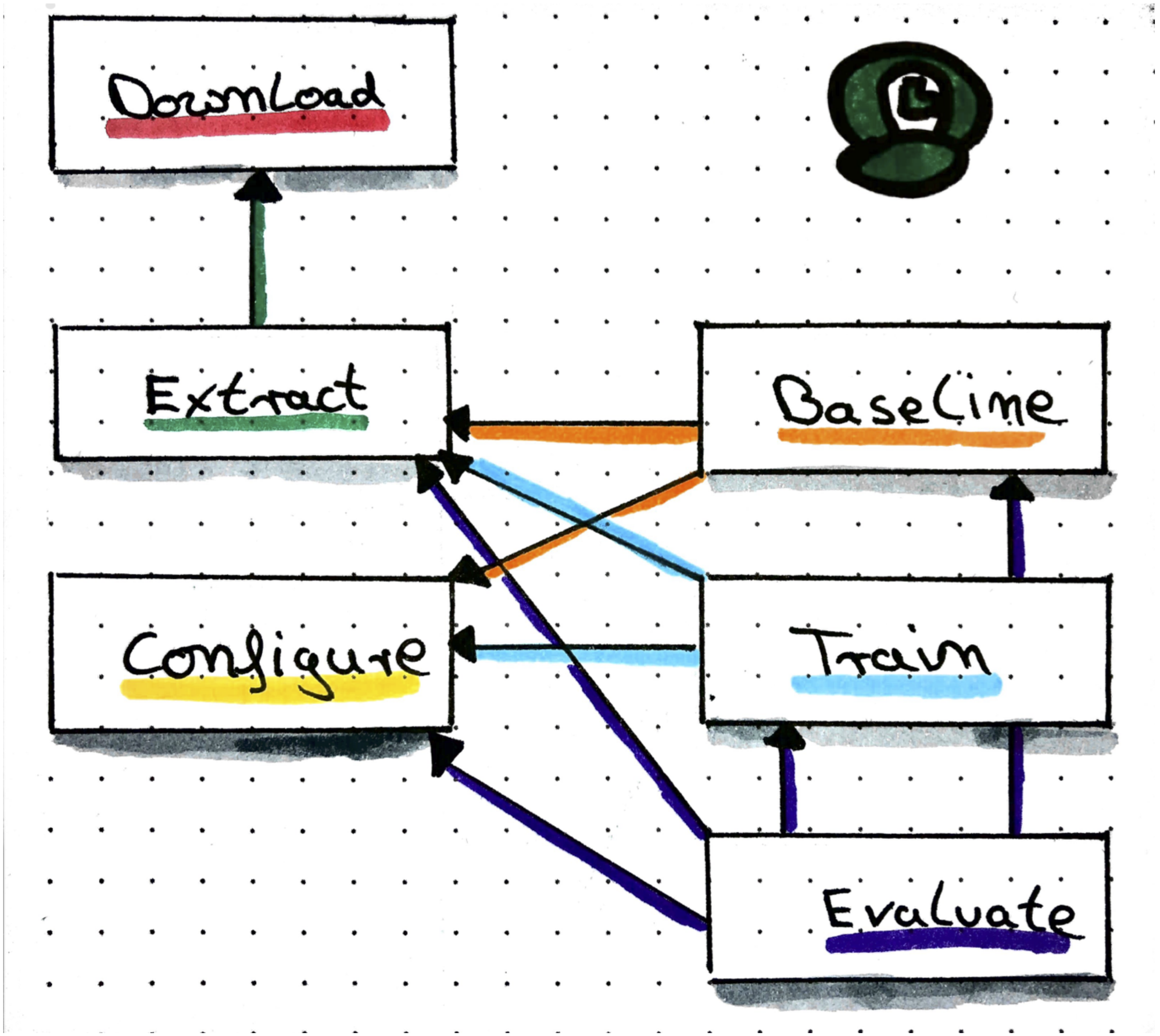
- like constructur parameters
- Luigi takes care of parameter validation
- Again, a lot of implementations already exist in the Luigi framework
- IntParameter, BoolParameter, DateParameter, etc...

Parameters

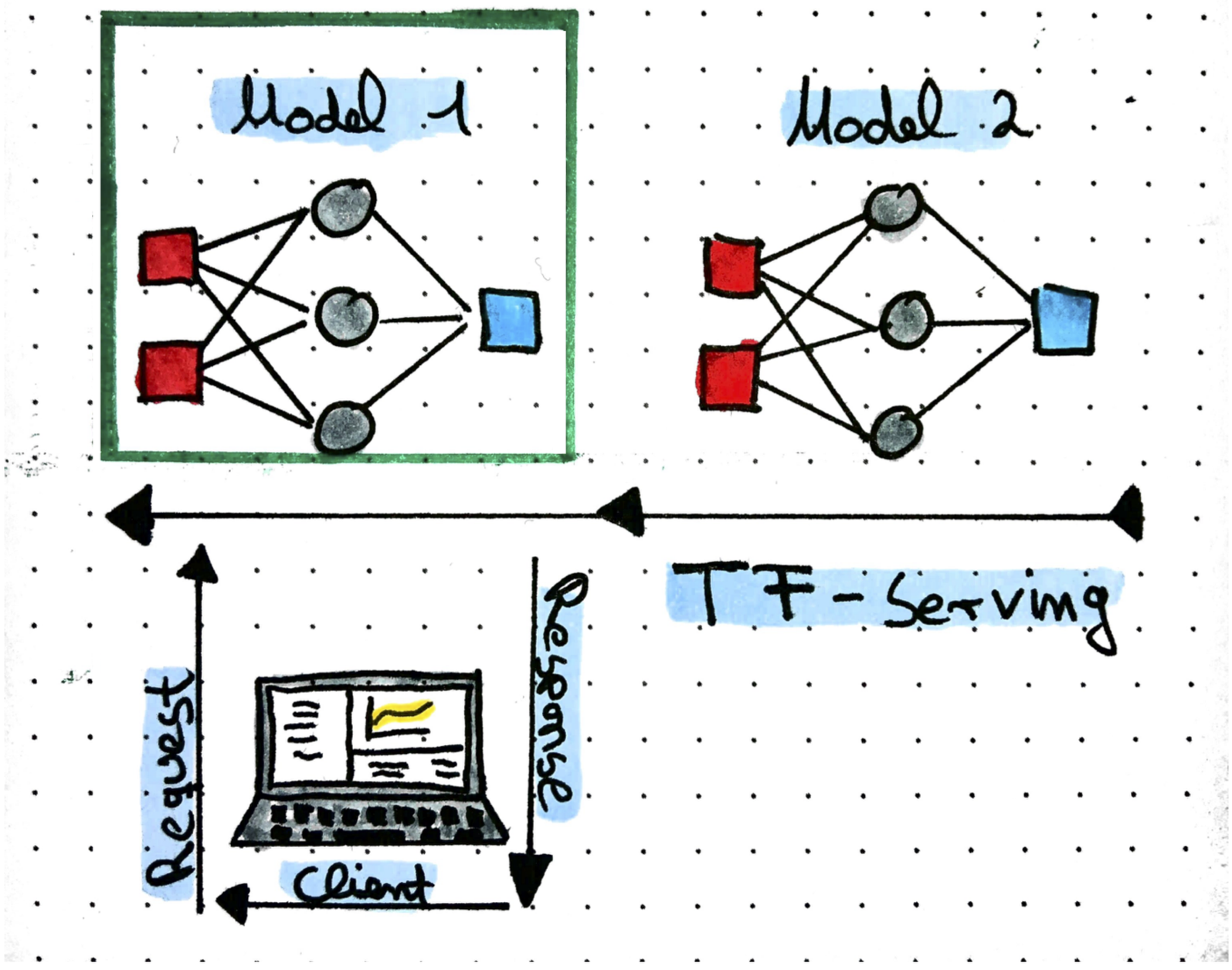

"../{name}/{version}/model.h5"
.h5
filepath
Exists
name: Param
version: IntParam
name = "cnn"
version = 1

How would our Workflow look like?

- Download the dataset
- Extract the data
- Create a preprocessing configuration
- Run the baseline validation
- Train the model
- Evaluate the model


DownLoad
Extract
BaseLine
Configure
Train
Evaluate

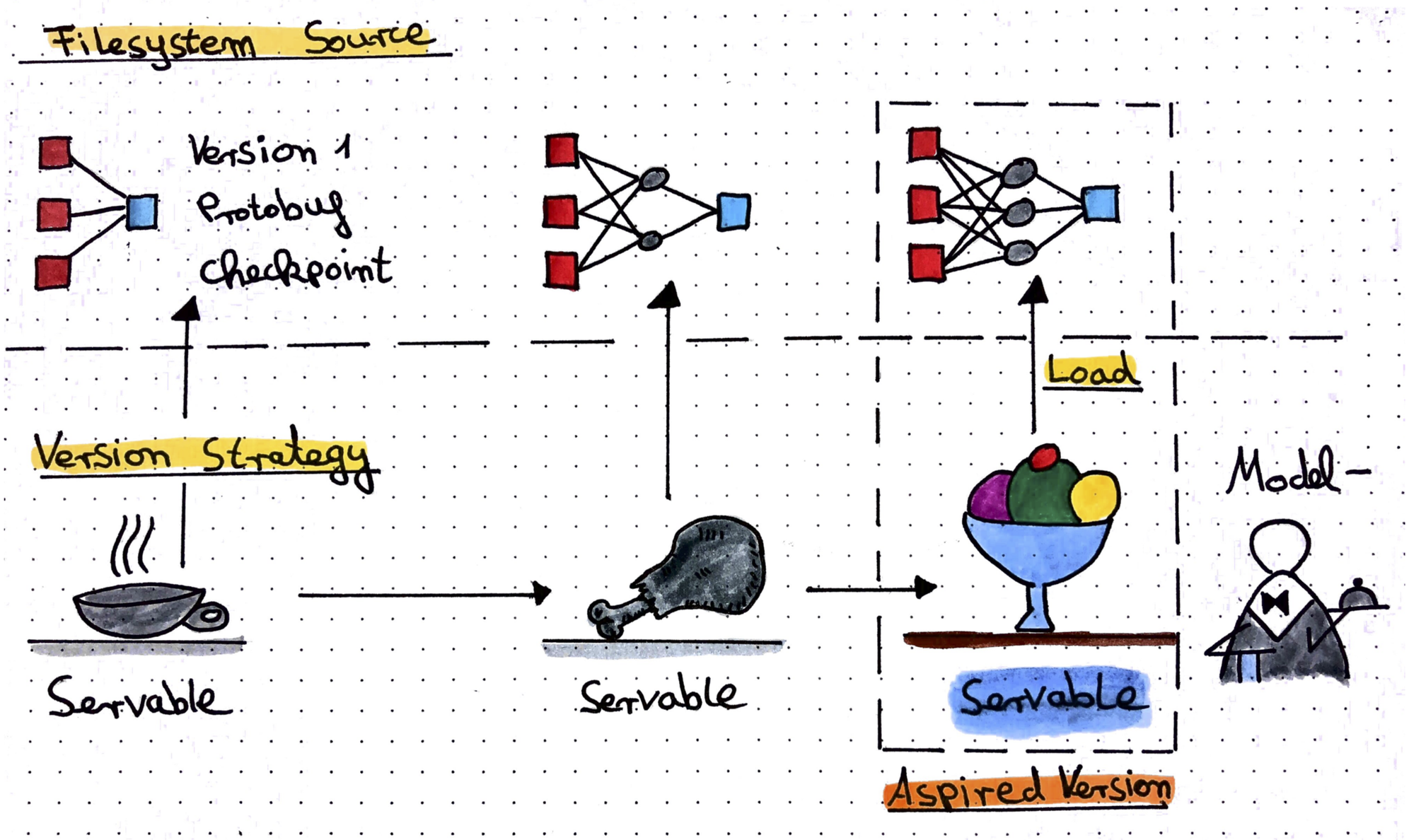
TensorFlow Serving



TensorFlow Serving

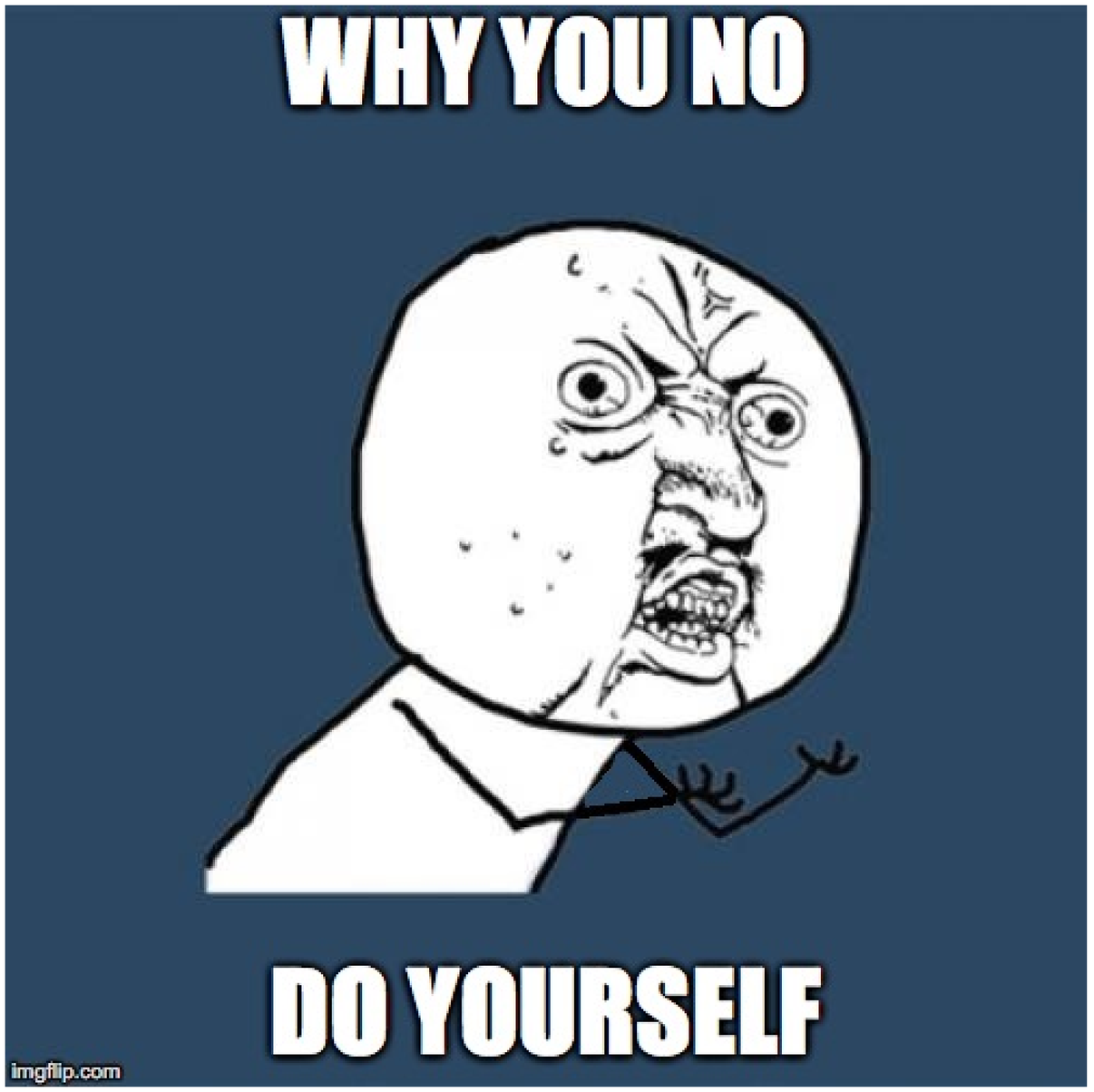
- ModelServer as Server-runtime for ML models
- Can natively handle TensorFlow graphs
- Highly flexible
- Designed for production use

TensorFlow Serving in Detail



TensorFlow Serving Components

- Protobuf file for graph, checkpoint files for weights (if not frozen)
- Version strategy defines how many models are loaded into memory
- Clients request inferences via gRPC or REST (yes REST!)
- TF-Serving consists of pluggable components, namely sources, loaders and version strategies


WHY YOU NO
DO YOURSELF
imgflip.com

Integrate pre-processing into model

- The pre-processing and the model are tightly coupled
- Datagenerator can be pickled to make training reproducable
- But how can the client be sure to do the "right" thing?

Possible ways:

- The client just "knows"
- Wrap service around model and expose an API
- Built the pre-processing right into the model

In [50]:
```
def pre_process(x):
    resized = tf.image.resize_images(x, size=[100, 100])
    max_color = tf.constant(255, dtype=tf.float32)
    rescaled = tf.divide(resized, max_color)
    return rescaled

additional = Lambda(pre_process, output_shape=(100, 100, 3), name="preprocessing")
```

In [51]:
```
prod_model = Sequential()
prod_model.add(InputLayer(input_shape=(None, None, 3)))
prod_model.add(additional)
for layer in model.layers:
    prod_model.add(layer)

prod_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
preprocessing (Lambda)       (None, 100, 100, 3)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 98, 98, 2)         56
_________________________________________________________________
batch_normalization_5 (Batch (None, 98, 98, 2)         8
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 49, 49, 2)         0
_________________________________________________________________
dropout_9 (Dropout)          (None, 49, 49, 2)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 4802)              0
_________________________________________________________________
dense_9 (Dense)              (None, 16)                76848
_________________________________________________________________
dropout_10 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 34
=================================================================
Total params: 76,946
Trainable params: 76,942
Non-trainable params: 4
_________________________________________________________________
```