Deploying Machine Learning Models in Python

Albert Au Yeung

PyCon HK 2018
23th November, 2018

Deploying Machine Learning Models in Python

Albert Au Yeung

PyCon HK 2018
23th November, 2018

Deploying Machine Learning Mode
in Python

Albert Au Yeung

PyCon HK 2018
23th November, 2018

What is this Talk About?

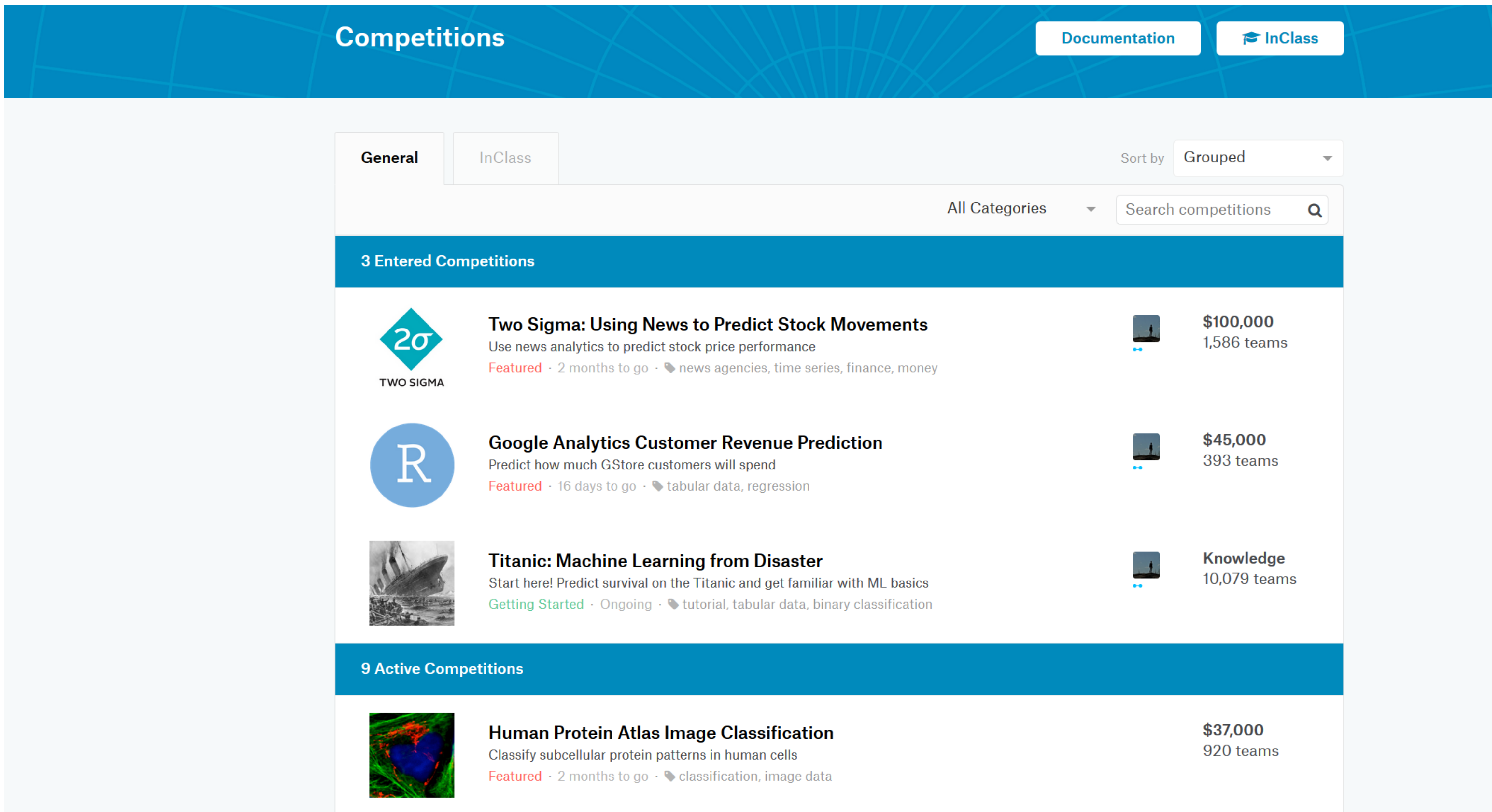
What is this Talk About?

- The need to **deploy** machine learning models
- Different kinds of **workflows**
- Common **strategies**
- **Options** in Python
- Other **considerations**
- NOT about using docker/kubernetes

The "Kaggle" Way of Machine Learning

The "Kaggle" Way of Machine Learning

- Kaggle: a Website that host machine learning competitions
- You train machine learning models and generate predictions on the
- Everything is done **offline**



Machine Learning Applications

Machine Learning Applications

- In practice, generating predictions is only a **small part** of a machin
- Consider a system that uses machine learning to recognize hand-w

Common ML Systems Workflow (1)

- Train offline ➤ Predict offline ➤ Store predictions in DB
- Example: **Recommender systems**
  - A model is trained **offline**
  - For each user, generate (pre-compute) a list of recommended items, store in **database**
  - When the user visits the Website, return the list of items

Common ML Systems Workflow

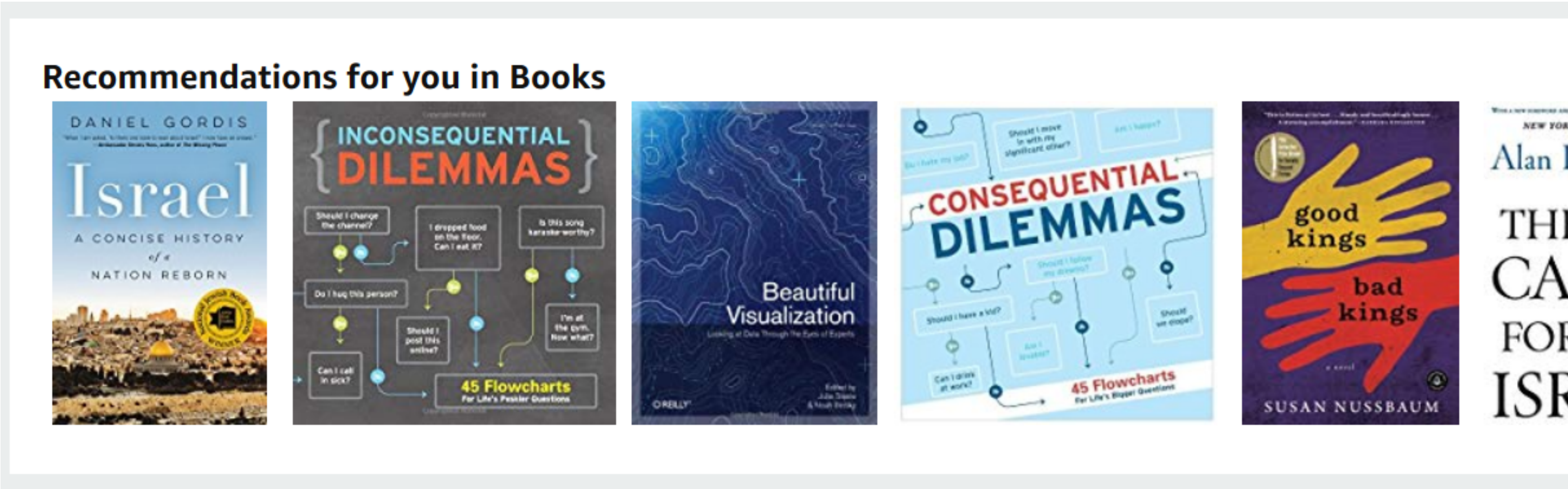
Common ML Systems Workflow

- There are several common **workflows** for machine learning system
  1. Train offline ➤ Predict offline ➤ Store predictions in DB
  2. Train offline ➤ Embed model in a device ➤ Predict online
  3. Train offline ➤ Make model available as a service ➤ Predict online
- Notes:
  - **Offline**
    separate from a production system; does not have to be completed in real time
  - **Online**
    part of a production system; perform tasks in real time

Common ML Systems Workflow (1)

Common ML Systems Workflow (1)

- Train offline ➤ Predict offline ➤ Store predictions in DB
- Example: **Recommender systems**
    - A model is trained **offline**
    - For each user, generate (pre-compute) a list of recommended items, store in **data**
    - When the user visits the Website, return the list of items



Common ML Systems Workflow (2)

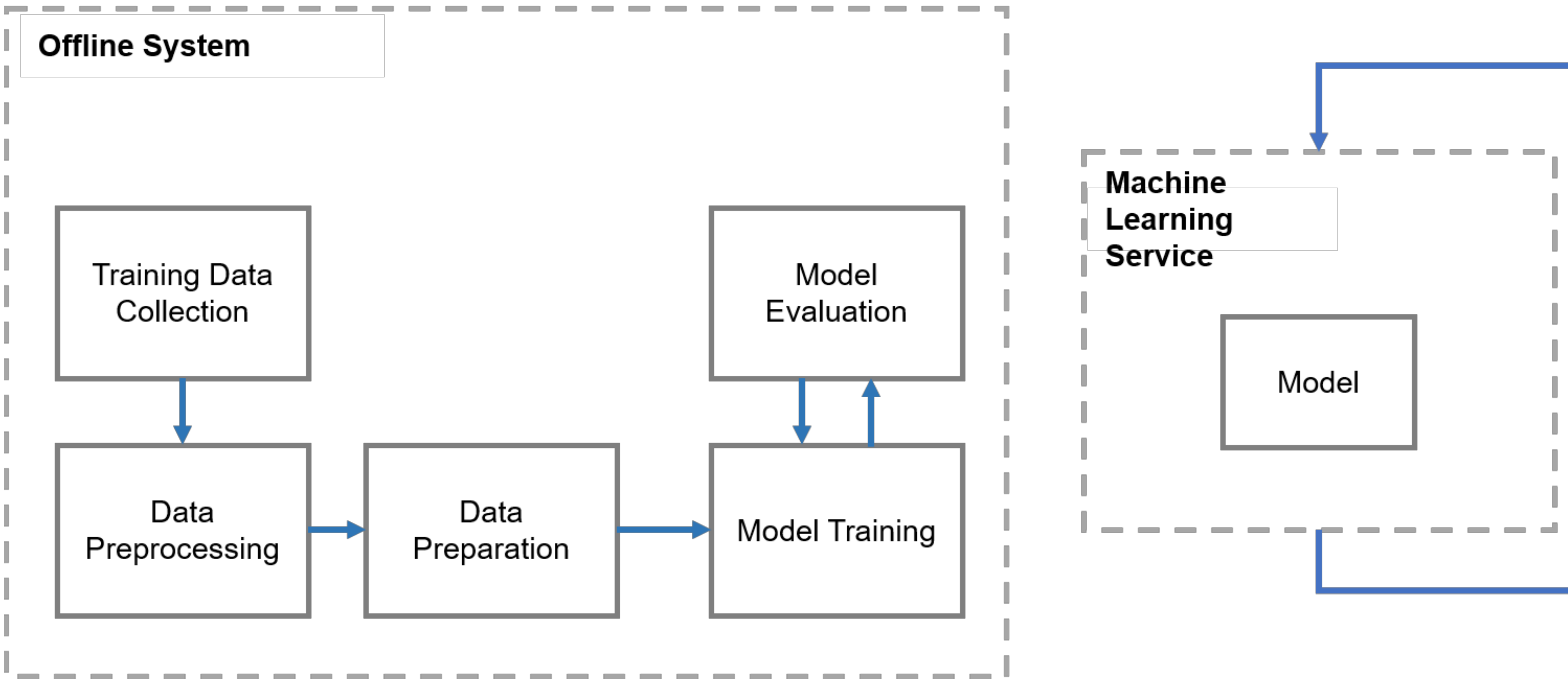
Common ML Systems Workflow (2)

- Train offline ➤ Embed model in a device ➤ Predict online
- Example: **Object detection using a drone**
  - A model is trained offline
  - The model together with other processing logic are downloaded to the drone's computer
  - The drone detects objects while it is in operation

Common ML Systems Workflow (3)

Common ML Systems Workflow (3)

- Train offline ➤ Make model available as a service ➤ Predict online
- Example: **Spam E-mail detection**
  - A classifier is trained offline with spam and non-span emails
  - Deployed as a service to serve users or other components in the system



Common ML Systems Workflow

Common ML Systems Workflow

- In (2) and (3), we need to think about how to **deploy** a machine lea
- Definition of **deploy**:

> *To place some resources into a position so as to be ready*
> *or use*

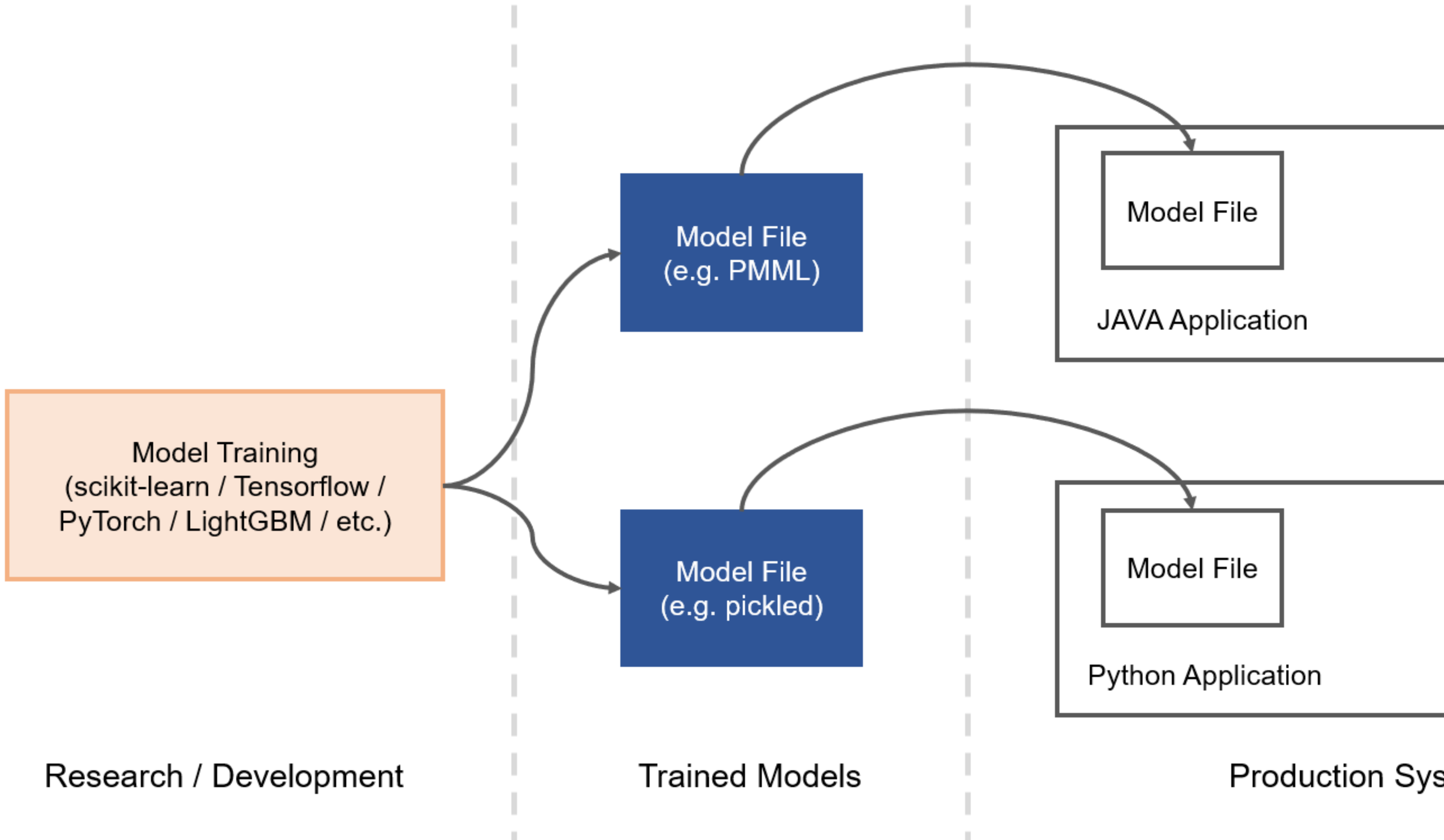
- In this talk, we will focus on **Use Case (3)**
- How to make machine learning models **available** to other users/sy

Common Strategies

Common Strategies

- **Persist** model in a **standard** format
- **Different languages** for development and production

- Serve models in r
- The **same** langua
used in developn



Persist Models in a Standard Format

Persist Models in a Standard Format

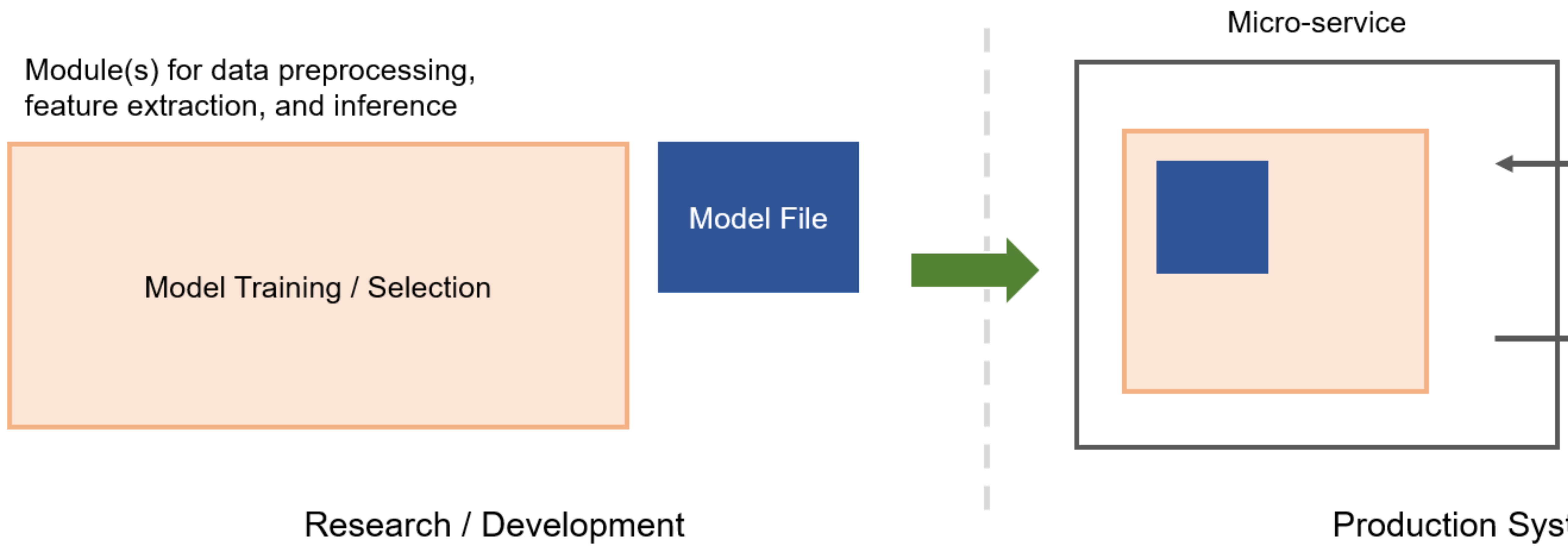
```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn2pmml import sklearn2pmml
from sklearn2pmml.pipeline import PMMLPipeline

# Load data ...

# Create pipeline and fit model
pipeline = PMMLPipeline([
  ("vec", CountVectorizer()),
  ("clf", LogisticRegression())
])
pipeline.fit(X, y)

# Write model in PMML format
sklearn2pmml(pipeline, "model.pmml", with_repr=True)
```

- **Predict**
  Langua
  - XML-b
    mode
  - scikit-
    sklear
  - Light(
    jpmm
- Referen
  Scikit-Le

Serve Models in Micro-services

Serve Models in Micro-services

- Models are persisted in a certain format specific to the language in
  (e.g. using `sklearn.externals.joblib`)
- The model (or even the module to load, pre-process and generate p
  wrapped in a **micro-service** that expose **endpoints** to receive requ



Options in Python

Options in Python

- XML-RPC
- HTTP REST micro-services
  - Flask: http://flask.pocoo.org/
  - Bottle: https://bottlepy.org/
  - Falcon: https://falconframework.org/
  - Vibora: https://github.com/vibora-io/vibora
  - AIOHTTP: https://aiohttp.readthedocs.io/en/stable/
- Asynchronous Messaging
  - Kafka: https://kafka.apache.org/
  - RabbitMQ: https://www.rabbitmq.com/
  - Redis: https://redis.io/

Wrapping Model Prediction in a Class

Wrapping Model Prediction in a Class

- Generating predictions can involve **pre-processing** and **post-proce**
- More convenient if everything is wrapped inside a **class**

```
class TextClassifier(object):
    """A Class wrapping the ML model"""

    def __init__(self):
        # Load the persisted model into memory
        self.model = joblib.load("model.pkl")

    def train(self):
        # Model training
        # ...

    def predict(self, x):
        y = self.model.predict([x])
        return y[0]
```

Tensorflow Models

Tensorflow Models

- For Tensorflow models, we need to keep a reference to the graph d

```
import tensorflow as tf
from keras.applications.resnet50 import ResNet50

class ImageClassifier(object):

    def __init__(self):
        self.graph = tf.get_default_graph()
        self.model = ResNet50(weights='imagenet')

    def preprocess(self, img):
        # preprocess the input image
        # ...

    def predict(self, img):
        x = self.preprocess(img)
        with self.graph.as_default():
            predictions = self.model.predict(x)
        ...
```

XML-RPC

XML-RPC

- XML-RPC: Remote procedure call based on HTTP and XML file forma
- The most straight-forward way if the **clients are also written in Py**

- Server

```
from xmlrpc.server import \
    SimpleXMLRPCServer
from model import TextClassifier

clf = TextClassifer()

address = ("localhost", 8000)
server = SimpleXMLRPCServer(address)
server.register_function(
    clf.predict, "predict")

server.serve_forever()
```

- Client

```
from xmlrpc.client imp

address = ("localhost"

# Create a proxy to t
with ServerProxy(addre
    server.predict("He
```

Flask

Flask

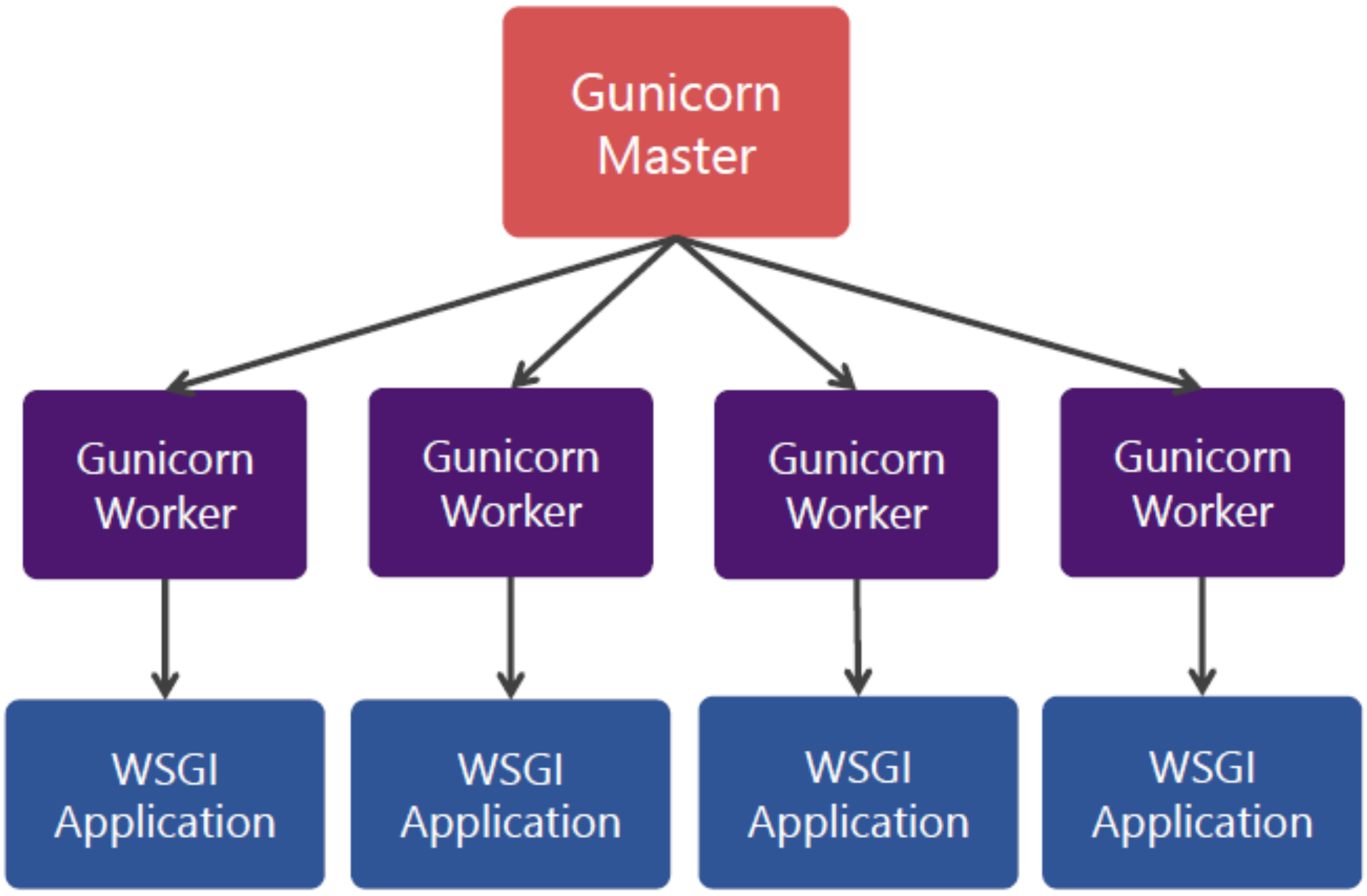
- A popular WSGI Web framework for creating HTTP APIs

```
from flask import Flask, current_app, request, jsonify
from model import TextClassifier

app = Flask(__name__)           # Create a Flask app
app.model = TextClassifier()  # Load model into the app

# Define the HTTP API for prediction
@app.route('/predict', methods=['POST'])
def predict():
    d = request.get_json()
    prediction = current_app.model.predict(d['x'])
    return jsonify(result=prediction)

# Start the Flask internal Web server
app.run()
```

Deploying

Deploying

- Flask is a **WSGI** Web Framework, it can be deployed using Gunicorn
- Gunicorn uses a **pre-fork worker** model (creates copies of your app
- `$ gunicorn app:app -b localhost:8000 -w 4`



Falcon

Falcon

- A Framework that focuses on REST APIs
- Falcon vs. Flask — Which one to pick to create a scalable deep learn

```
import falcon
from model import TextClassifier

class Handler(object):

    def __init__(self):
        self.model = TextClassifier()

    def on_post(self, req, resp):
        data = json.loads(req.stream.read())
        resp.media = {"result": self.model.predict(data['x'])}

app = falcon.API()
app.add_route('/predict', Handler())
app.run()
```

Vibora

Vibora

- Asynchronous HTTP client/server framework (requires Python 3.6+
- API very similar to Flask

```
from vibora import Vibora, Request
from vibora.responses import JsonResponse
from model import TextClassifier

app = Vibora()  # Create an Vibora application
app.add_component(TextClassifer())  # Add a globally available component

@app.route('/predict', methods=['POST'])
async def predict(request: Request):
    data = await request.json()
    model = app.get_component(TextClassifier)  # Get reference to the model
    prediction = model.predict(data['x'])
    return JsonResponse({"result": prediction})
```

AIOHTTP

AIOHTTP

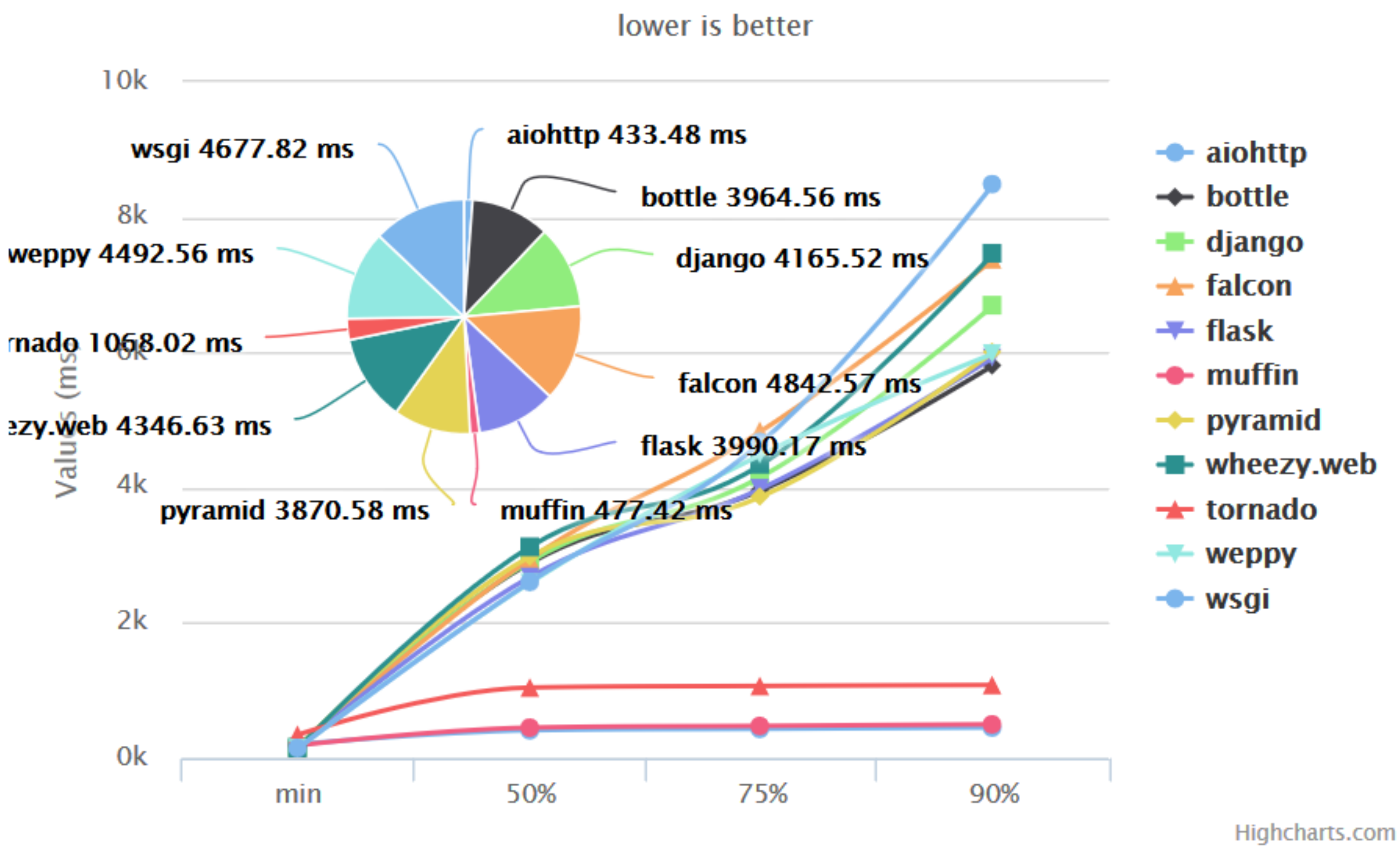
- Framework for implementing Asychronous HTTP client/server on t

```
from aiohttp import web
from model import TextClassifier

async def handle(request):
    data = await request.json()
    prediction = request.app['model'].predict(data['input'])
    return web.json_response({"result": prediction})

app = web.Application()
app.router.add_post('/predict', handle)
app["model"] = TextClassifier()

web.run_app(app, host='localhost', port=8000)
```

Comparing Web Frameworks

Comparing Web Frameworks

- Some benchmarking results can be found here or here
- For ML services, the choice of framework seems not too important,
  the time will be spent on data **processing** and **inference**



Simple request and response
http://klen.github.io/py-frameworks-bench/

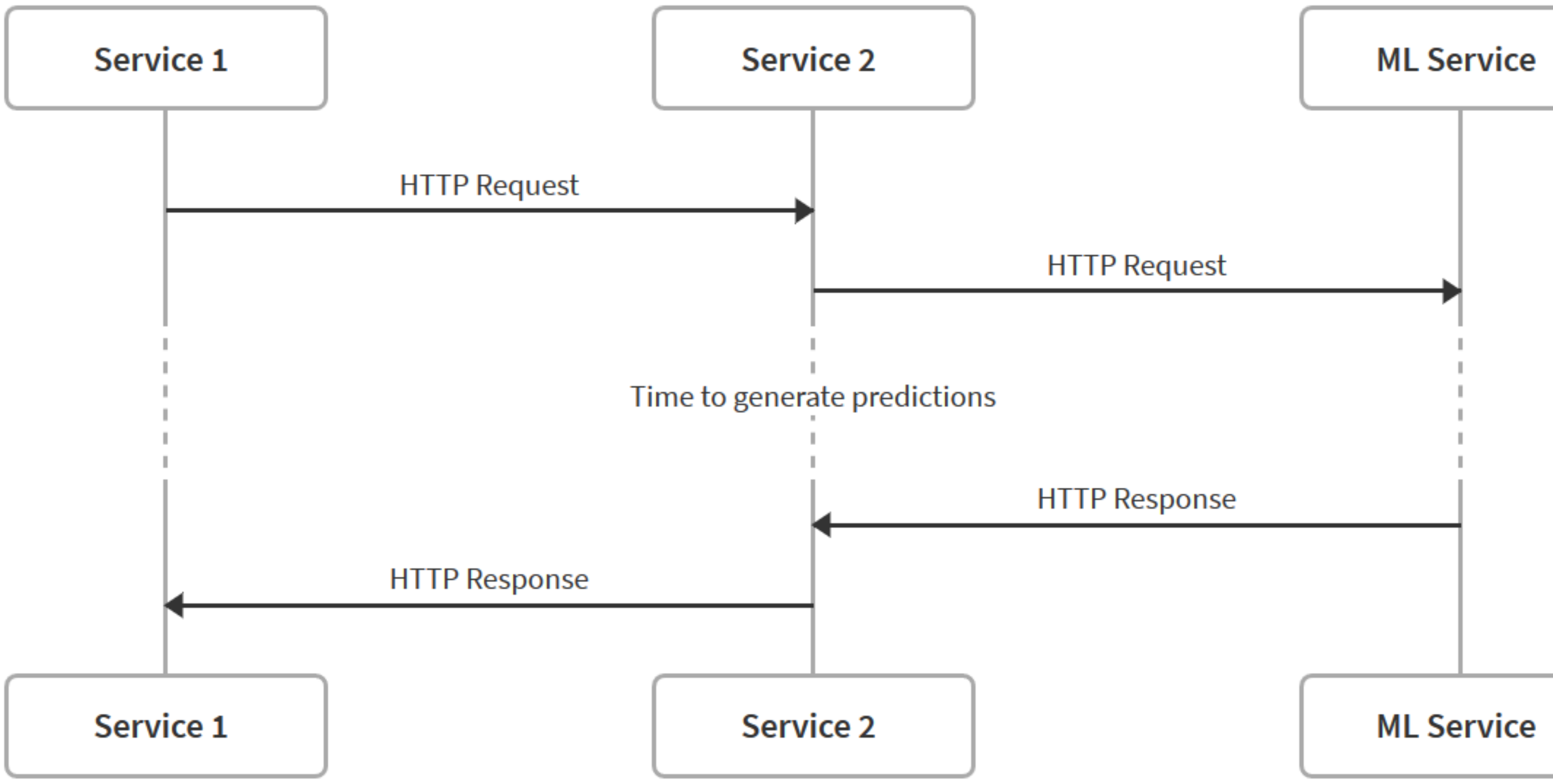
Frameworks	Requ
Tornado	1
Django	1
Flask	1
Aiohttp	2
Sanic	7
Vibora	13

POST JSON data
https://github.com/v

Disadvantage of Using HTTP APIs

Disadvantage of Using HTTP APIs

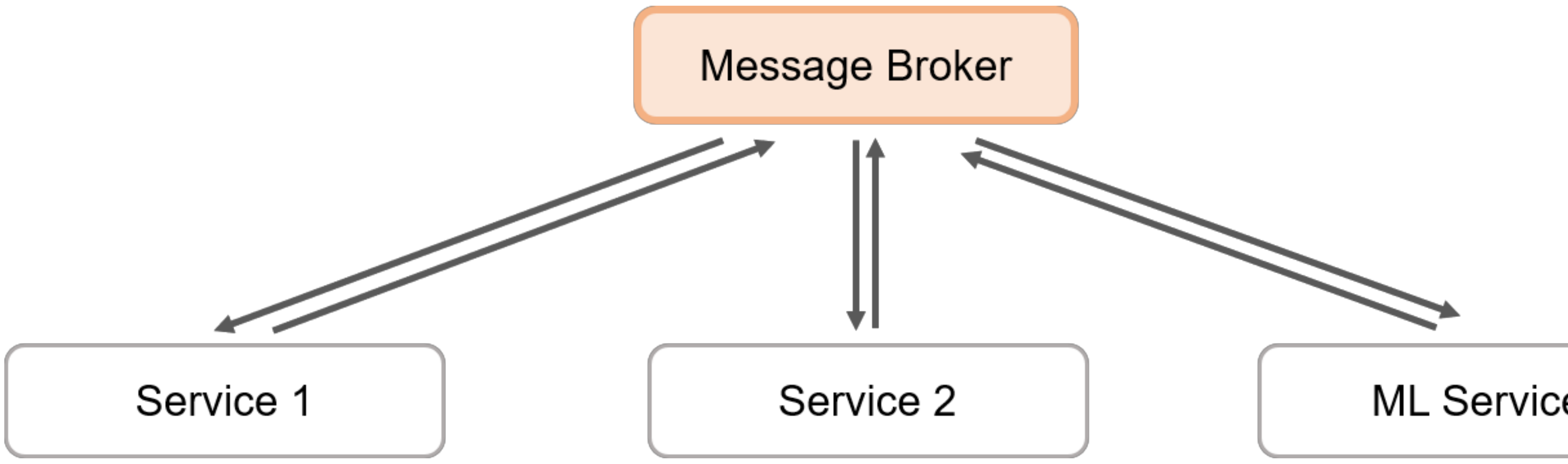
- HTTP REST APIs are simple to implement, however the process is **s**
- The client must **wait** until the ML service has finished the process o predictions
- In a complex system involving a lot of components, this may not be



Asynchronous Messaging

Asynchronous Messaging

- If components are relatively independent, using **asynchronous me** more efficient use of the services' resources
- Send requests and responses to a **message broker** instead of direc



- Options:
  - Redis https://redis.io/
  - RabbitMQ: https://www.rabbitmq.com/
  - Kafka: https://kafka.apache.org/

Rodis

Redis

- Redis is a key/value cache, but can also be used as a **message queu**

- Client Side

```
from redis import StrictRedis

# Get a connection to Redis
queue = StrictRedis(host='localhost',
                    port=6379)

# Create a message
message = "Hello!".encode("utf-8")

# Send the message to a channel
# named "prediction"
queue.publish("prediction", message)
```

- ML Service

```
from redis import Str
from model import Tex

queue = StrictRedis(ho
                    po
pubsub = queue.pubsub
pubsub.subscribe('pred

while True:
    x = p.get_message
    y_pred = model.pre
    # send result back
```

Kafka

Kafka

- A scalable and distributed message queue
- Two Python packages available: kafka-python and confluent-kafka

- Producer of messages

```
from kafka import KafkaProducer

# Create a message producer
address = 'localhost:1234'
producer = KafkaProducer(
    bootstrap_servers=address)

# Send message to a topic
producer.send('prediction', # topic
              '{"x": "Hello!"}') # msg
...
```

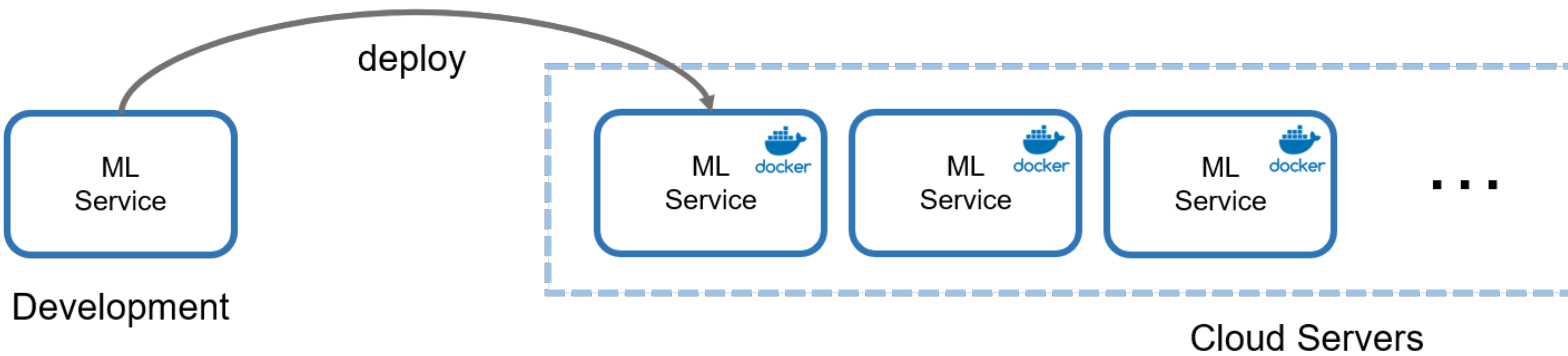
- Consumer of mes

```
from kafka import Kafk

# Create a consumer o
consumer = KafkaConsun

# Get message from bro
for msg in consumer:
    # Decode message
    content = msg.valu
    data = json.loads
    ...
```

Scaling ML Services

Scaling ML Services

- Nowadays micro-services are usually deployed as docker containe
systems such as Kubernetes
- **Scaling** involves creating multiple containers, each container runn
service
- Challenge: How to configure **resources allocated to each containe**



Scaling ML Services

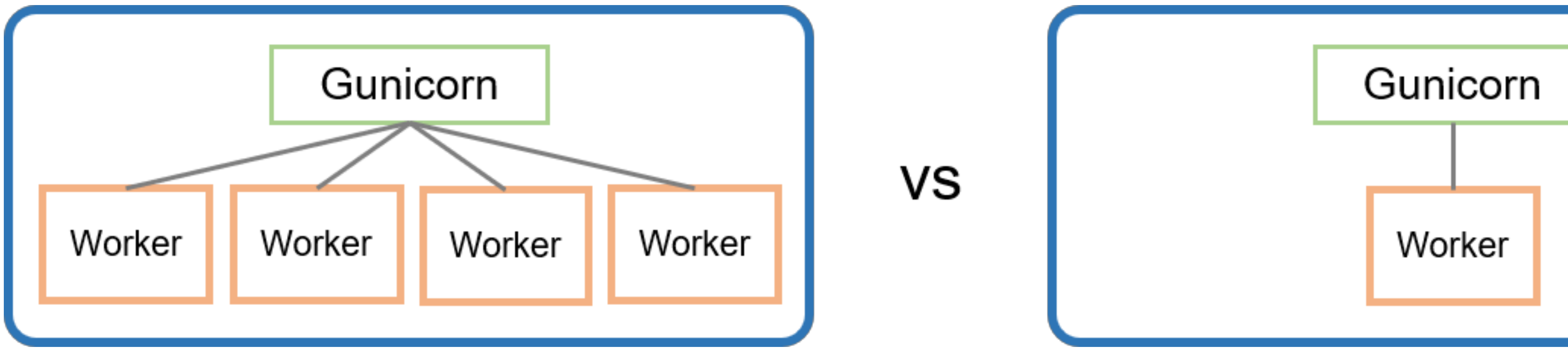
Scaling ML Services

- Some models can benefit from **multiple cores** using multi-threadin
  processing, e.g.:
  - scikit-learn's RandomForestClassifier
  - Facebook's fastText
  - Deep learning models built using Tensorflow or PyTorch
- Some models are huge and consume a lot of **RAM**
- Some models can only run on **single cores**
- Using GPUs, sometimes **batch processing** can be faster

Single vs. Multiple Workers

Single vs. Multiple Workers

- Example: using **Flask** and **Gunicorn** to deploy a Tensorflow model
- Tensorflow models can utilize **multiple cores** during inference
- Would be better to scale using multiple containers, each allocated
rather than having multiple workers in a single container



Summary

Summary

- When using a model in a **production system**, in addition to the per
(i.e. accuracy / precision / recall of the model, we need to consider:

1. **Model Size** ➤ would it be too big to be copied around?
2. **Memory** ➤ how much memory will it take up after loaded?
3. **CPU** ➤ how much CPU resources needed to generate a prediction?
4. **Time to Predict** ➤ time requried to generate a prediction
5. **Preprocessing** ➤ how complicated are the preprocessing steps?

Thank You!

http://www.albertauyeung.com
albertauyeung@gmail.com

Slides Avaliable at:
http://talks.albertauyeung.com/pycon2018-deploy-