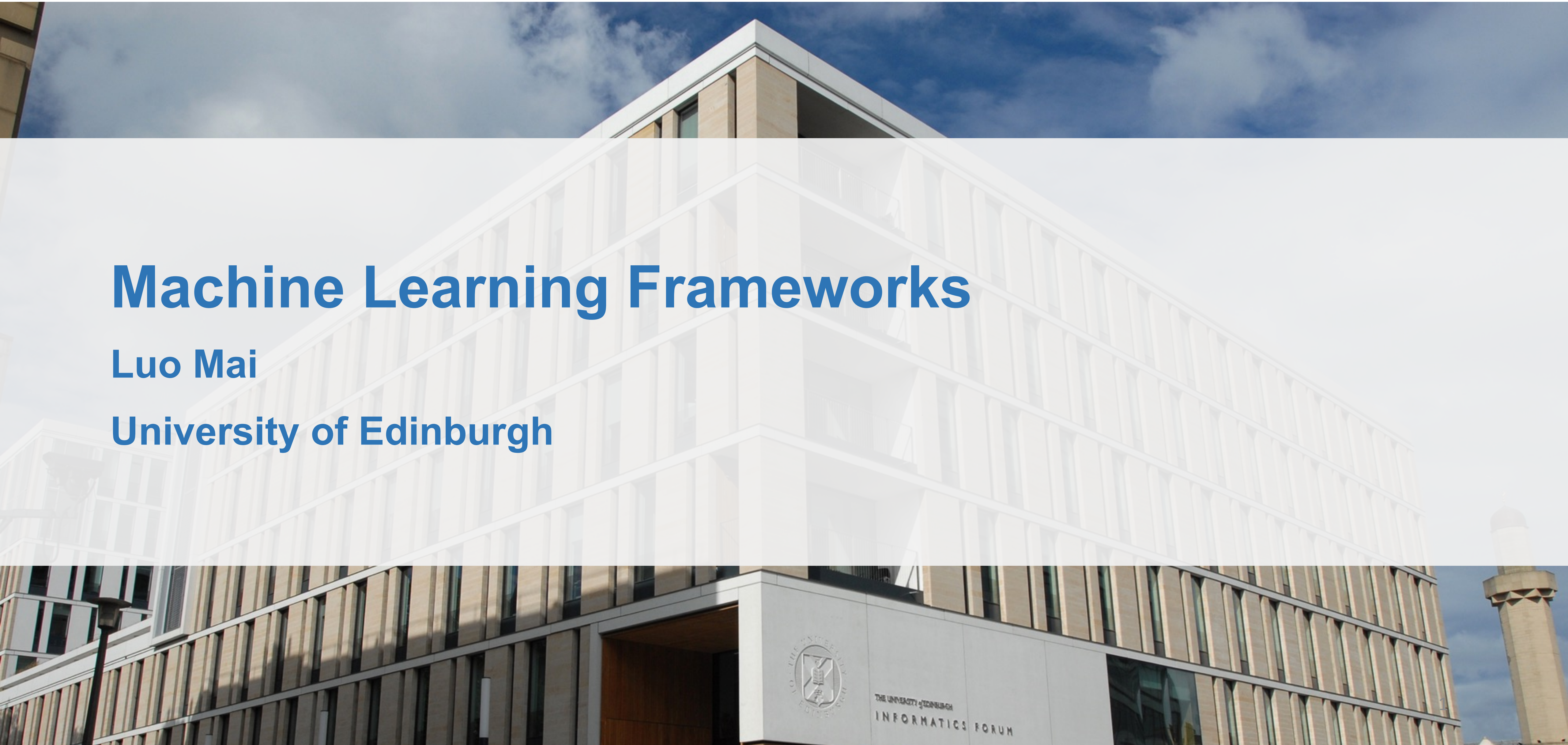
THE UNIVERSITY *of* EDINBURGH
informatics



Machine Learning Frameworks

Luo Mai

University of Edinburgh

Many ML applications are emerging

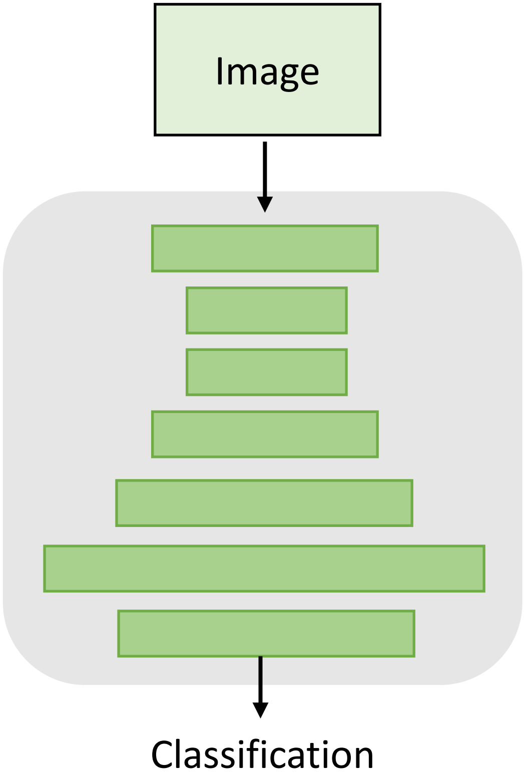
Deep neural networks

- Better accuracy
- High computation cost
- **Gradient-based** training

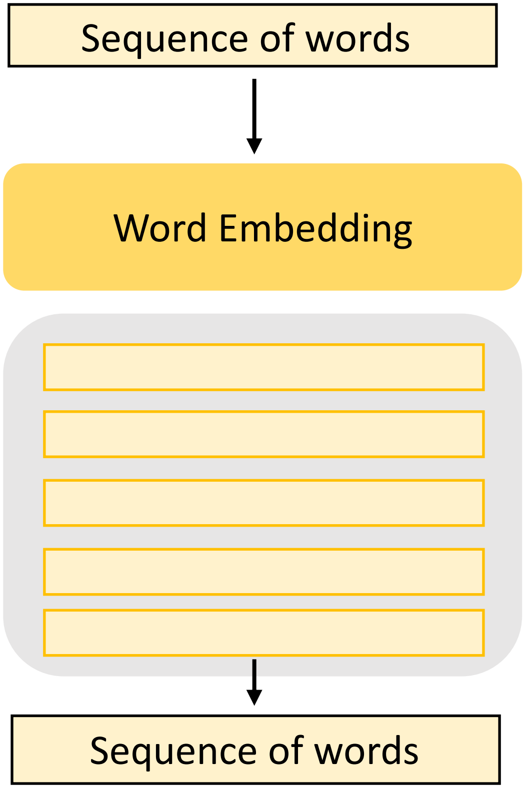
Diverse applications

- Natural language processing
- Deep reinforcement learning
- Graph neural networks
- …

Computer Vision



NLP



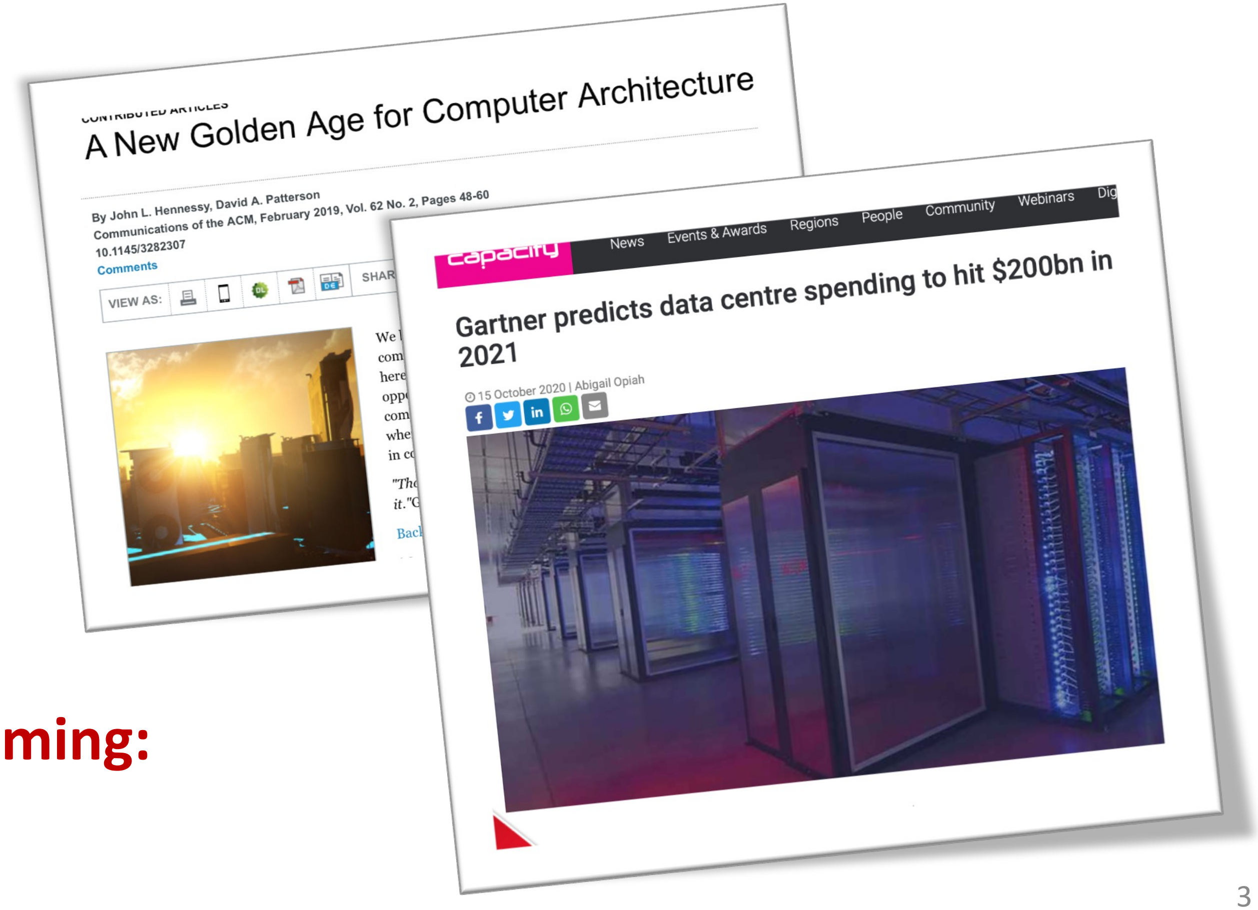
Massive computational power is available

Heterogeneous processors

- CPUs, GPUs and TPUs
- 10 – 100x acceleration

Global data centres

- Easy access to PB-scale data
- 100,000s machines



Three key factors that drive AI booming:
Algorithms, Hardware, Data

ML frameworks: A new category of system software

	Neural Networks	Automatic Differentiation	Un/semi-structured data management	Training & Inference	Heterogenous Processors	Distributed Execution
Neural network libraries (Theano, Caffe)	✓	✓	✗	✗	✓	✗
Data parallel systems (Spark, Giraph)	✗	✗	✗	✗	✗	✓
ML framework (PyTorch, TensorFlow)	✓	✓	✓	✓	✓	✓

THE UNIVERSITY of EDINBURGH
informatics

System architectures of ML frameworks

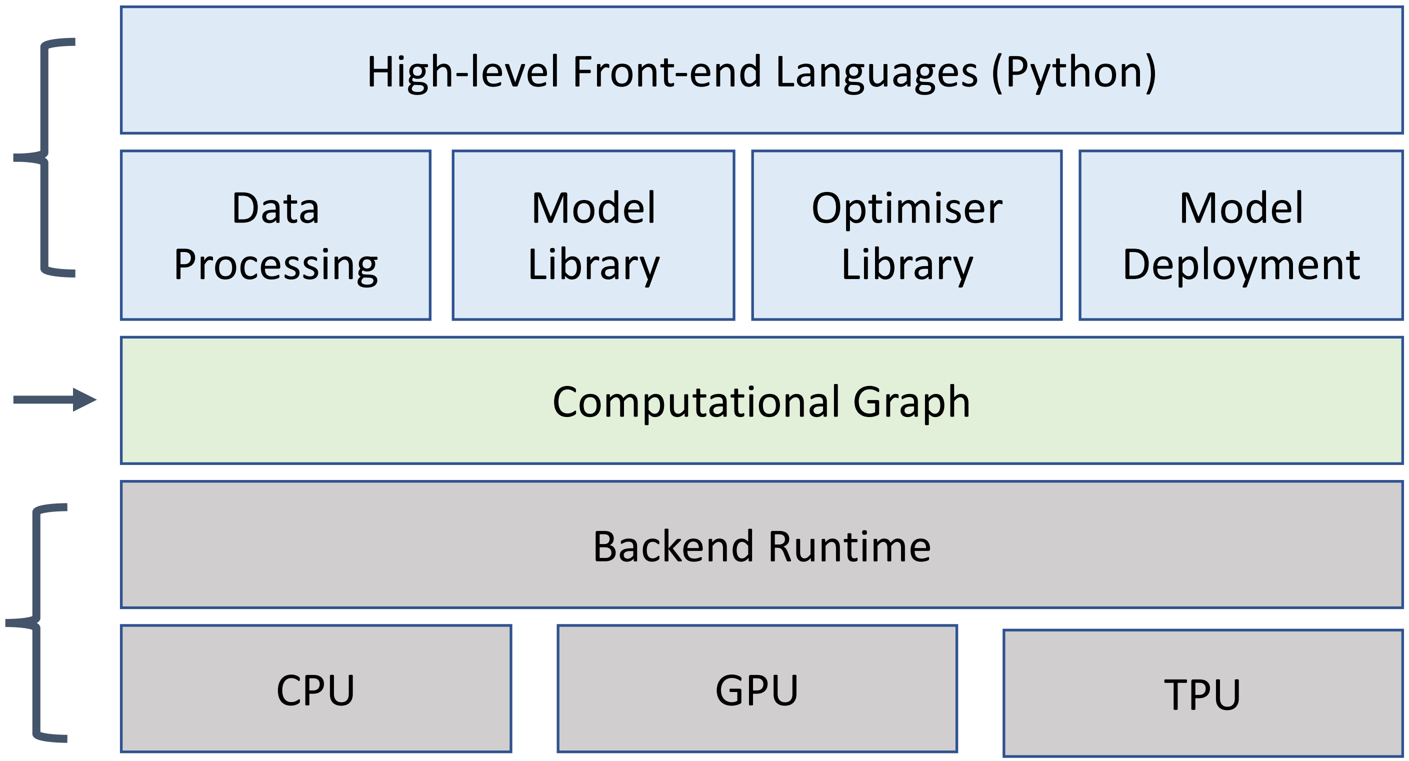
Design Goals

1. Programming abstraction: Supporting ML in different applications

2. Execution engine: Enable gradient-based computation & parallelise computation

3. Hardware runtime: utilise all heterogeneous processors

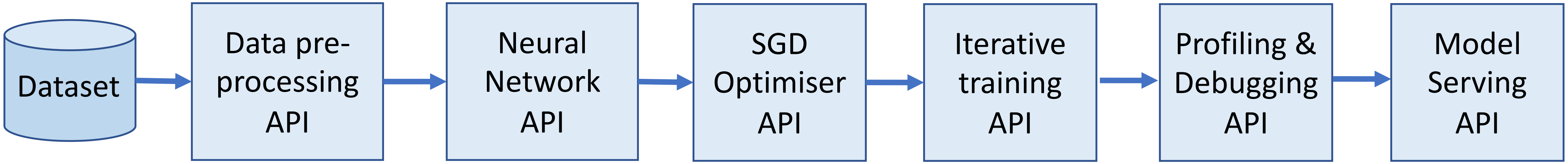
Architecture



THE UNIVERSITY of EDINBURGH informatics

ML framework programming abstraction

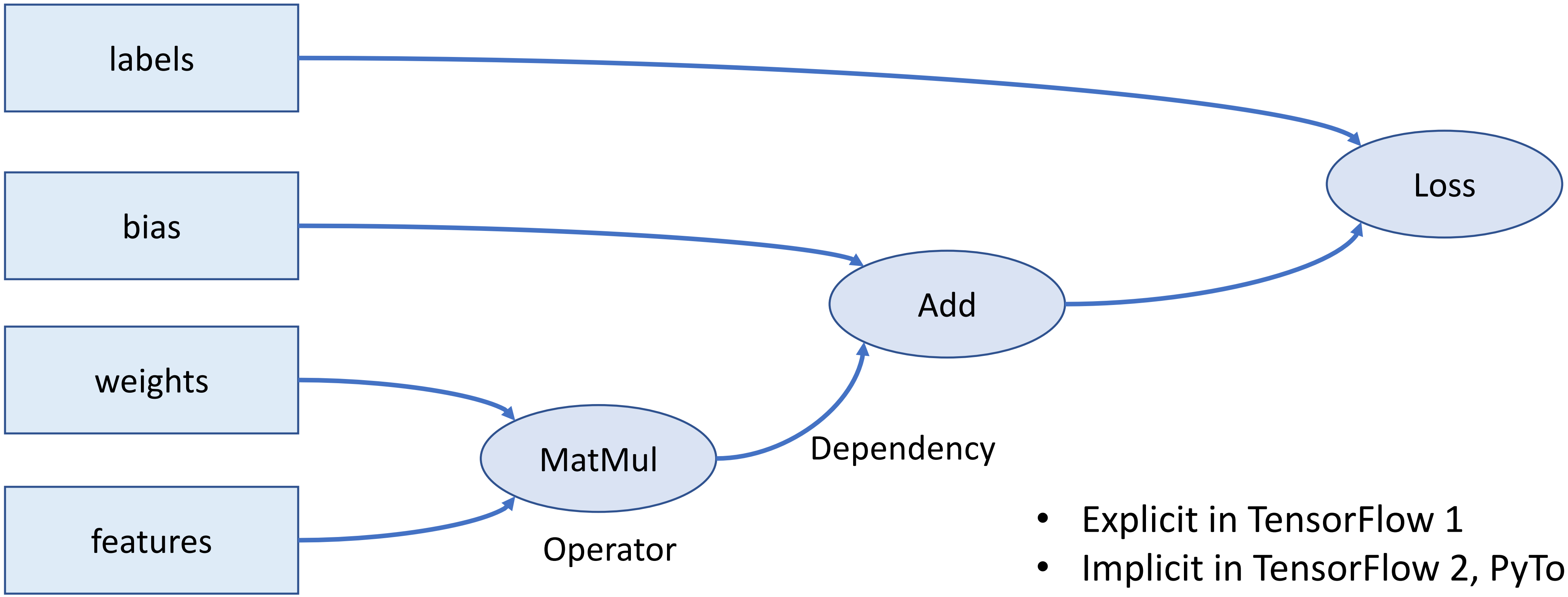
Typical ML Workflow



An unified programming abstraction for different ML applications (DNNs, GNNs, DRLs, …)

Questions?

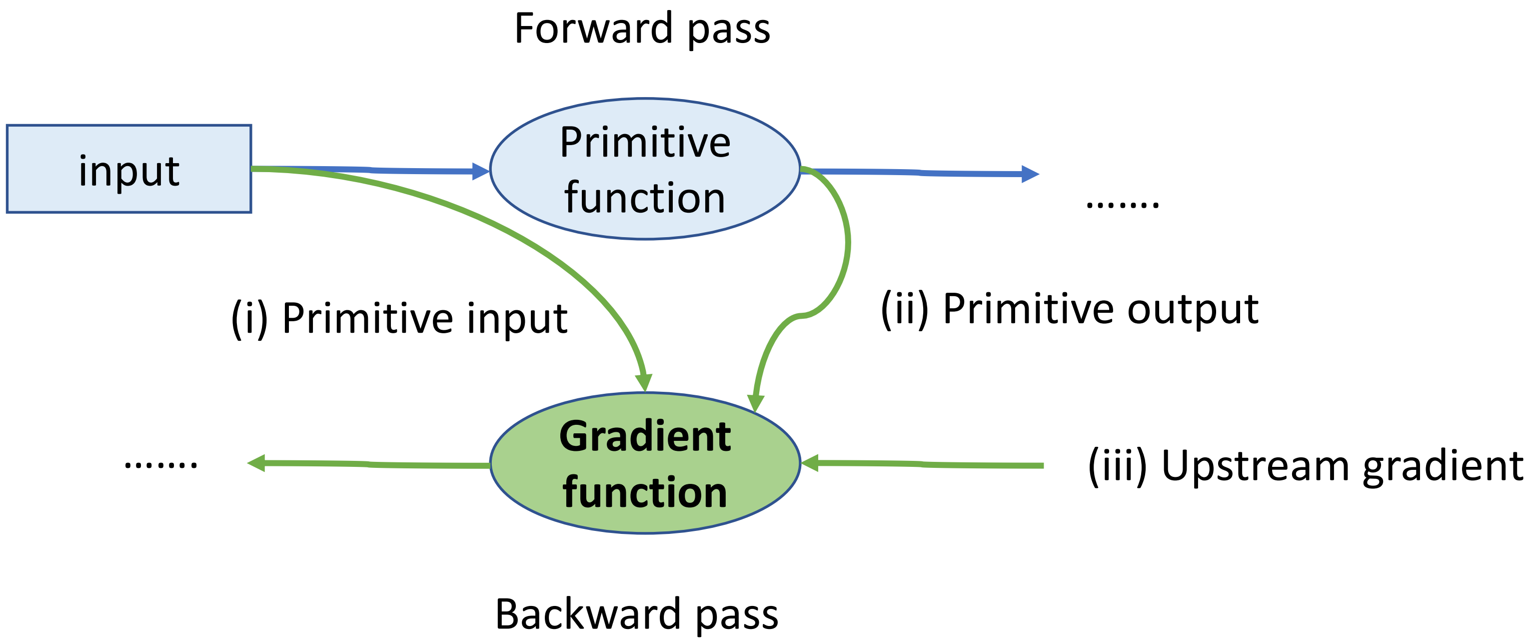
Expressing ML programs as computational graphs



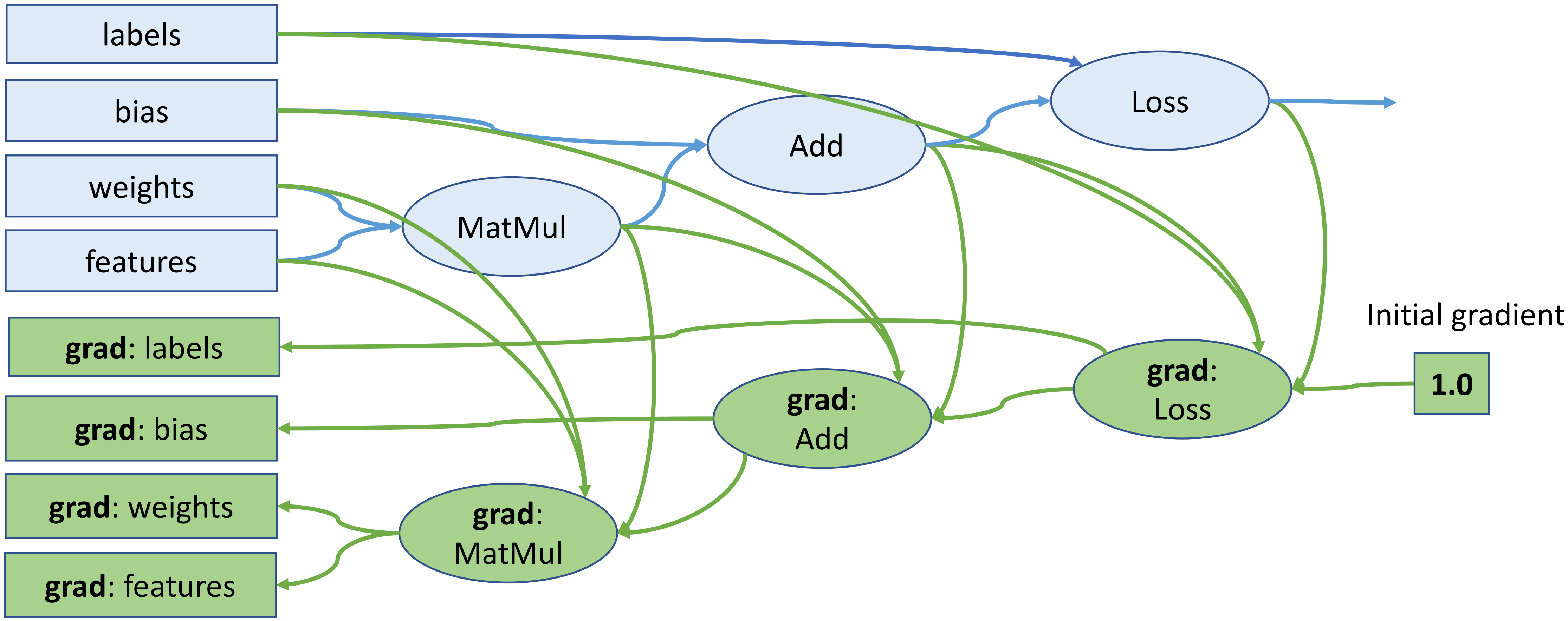
- Explicit in TensorFlow 1
- Implicit in TensorFlow 2, PyTorch

How to compute gradients automatically?

Automatic differentiation through the chain rule: Gradient function takes its primitive function (i) inputs and (ii) output as parameters along with the (iii) gradient of the function outputs with respect to the final outputs.



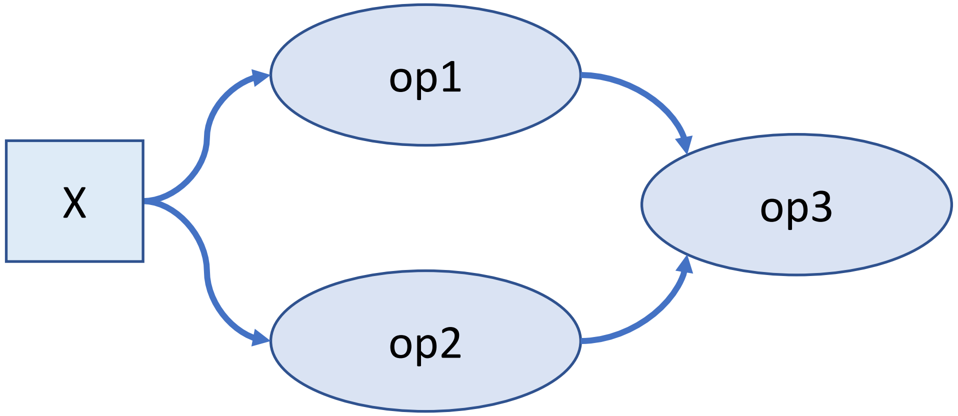
An automatic differentiation example



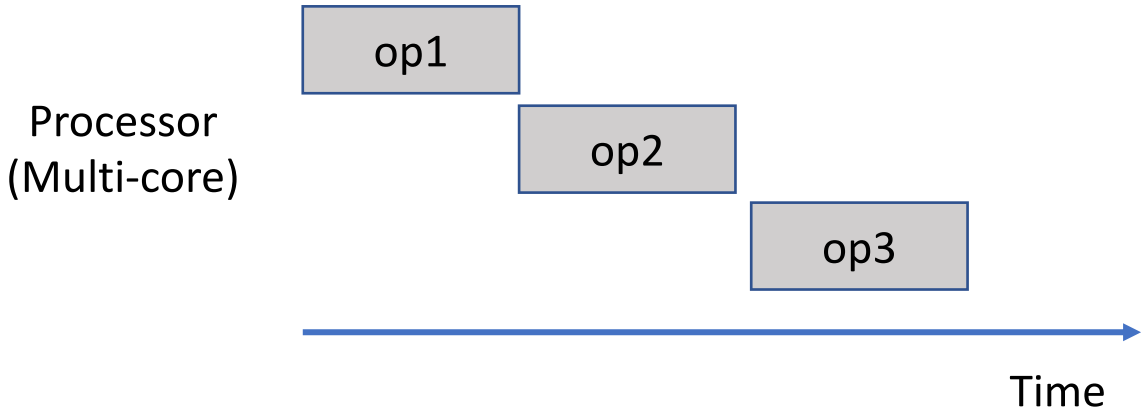
Discovering parallelism for better performance

```python
def model(x):
    y1 = op1(x)
    y2 = op2(x)
    return op3(y1, y2)
```

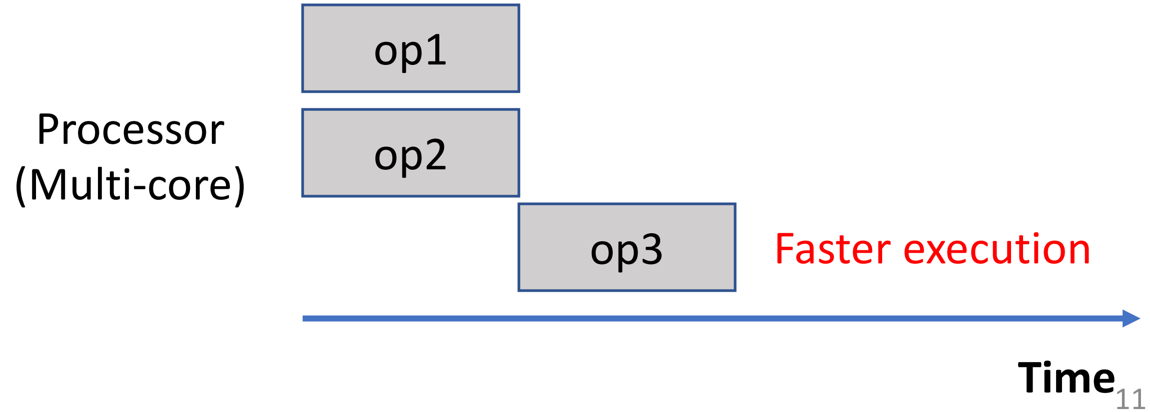
Equivalent Graph



Imperative Execution



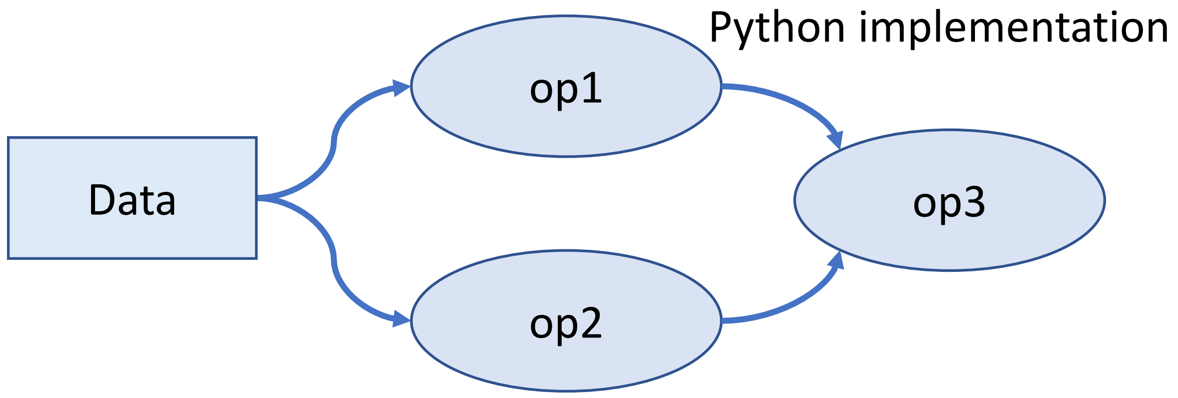
Parallel Execution



Questions?

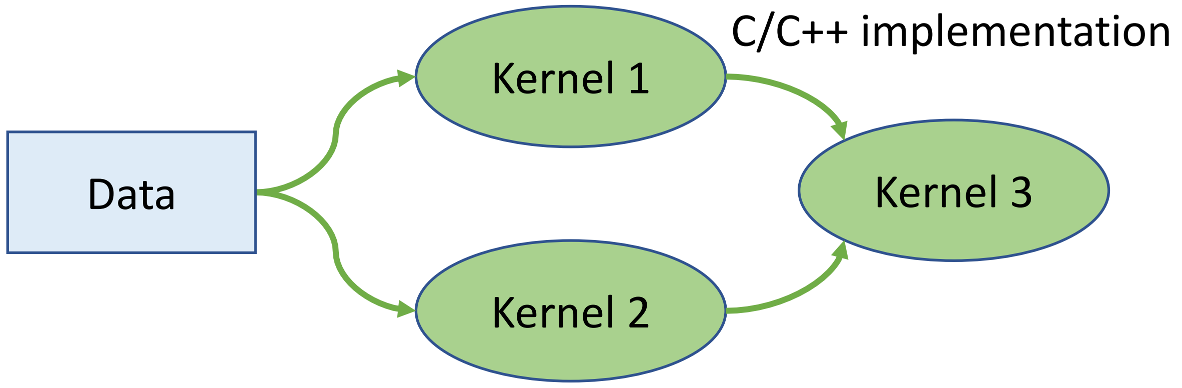
Frontend and backend languages

Front-end language: Python
- Simple and flexible
- Poor performance
- Global Interpreter Lock (GIL)



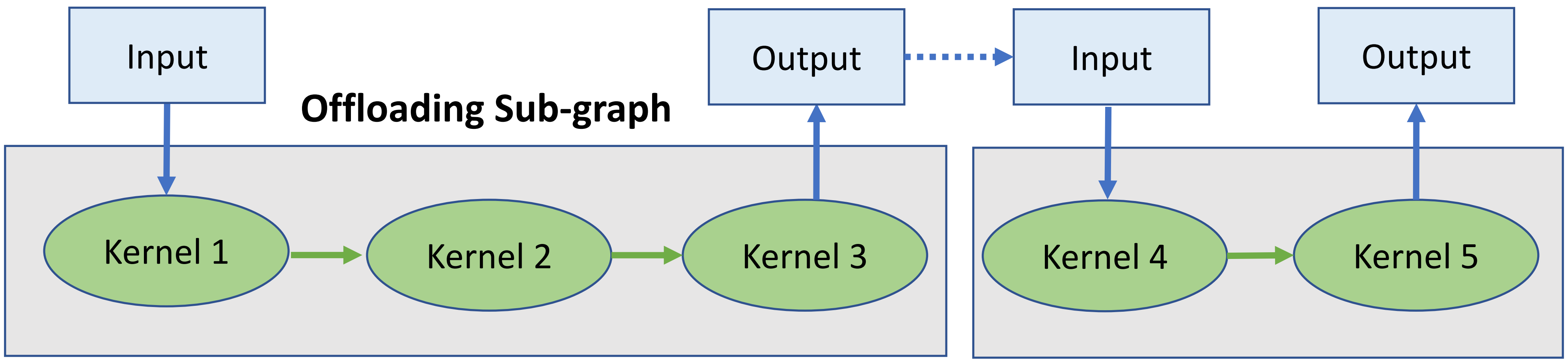
Equivalent Back-end Graph

Back-end language: C/C++
- Hardware-friendly
- Excellent performance



Offloading sub-graphs to heterogeneous processors

Problem: Frequently launching C++ kernels (e.g., system calls) in Python has large performance overhead



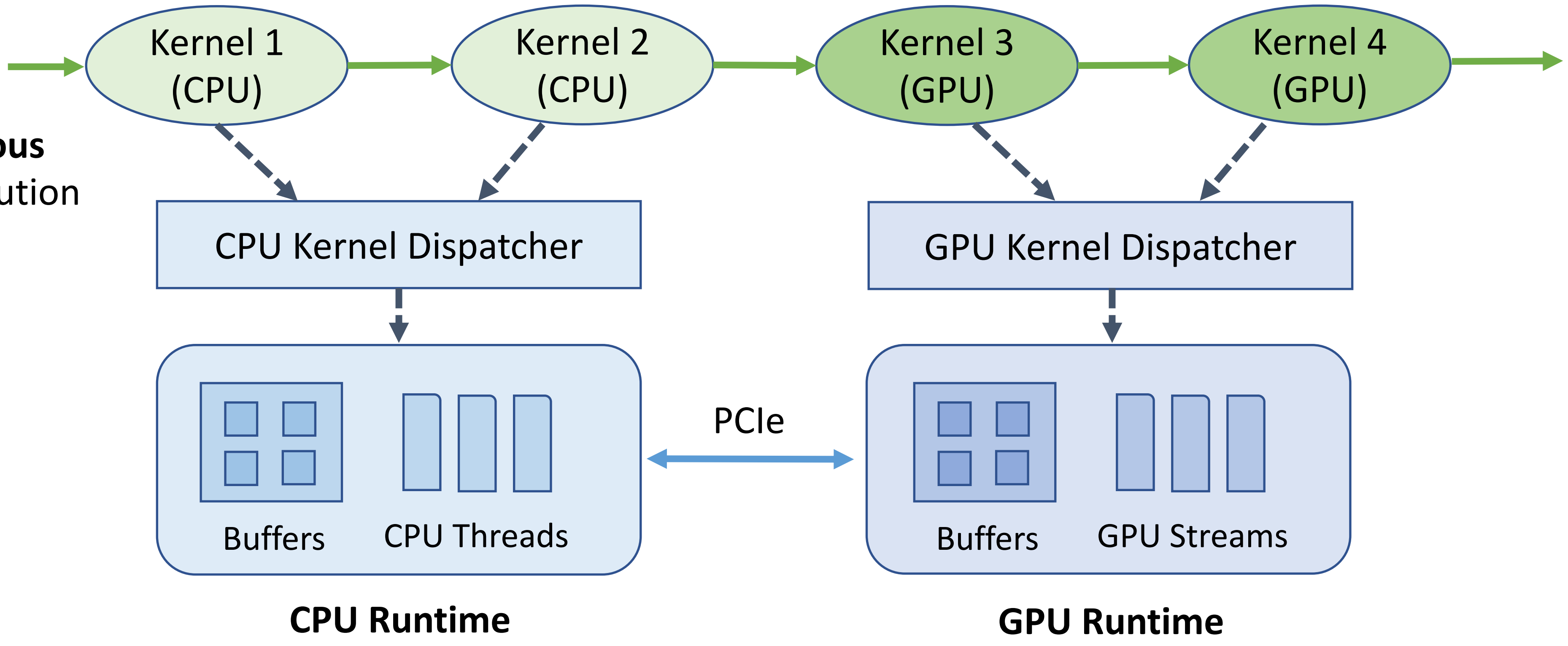
Discovering Sub-graph

- **User annotation** to discover sub-graphs: @tf.function (TensorFlow 2), @jit.script (PyTorch)
- **Just-in-Time (JIT)** compilation: @jit.trace (PyTorch)

Using heterogenous processors

Operators in ML models have execution kernels for CPUs and GPUs

Asynchronous kernel execution



Summary

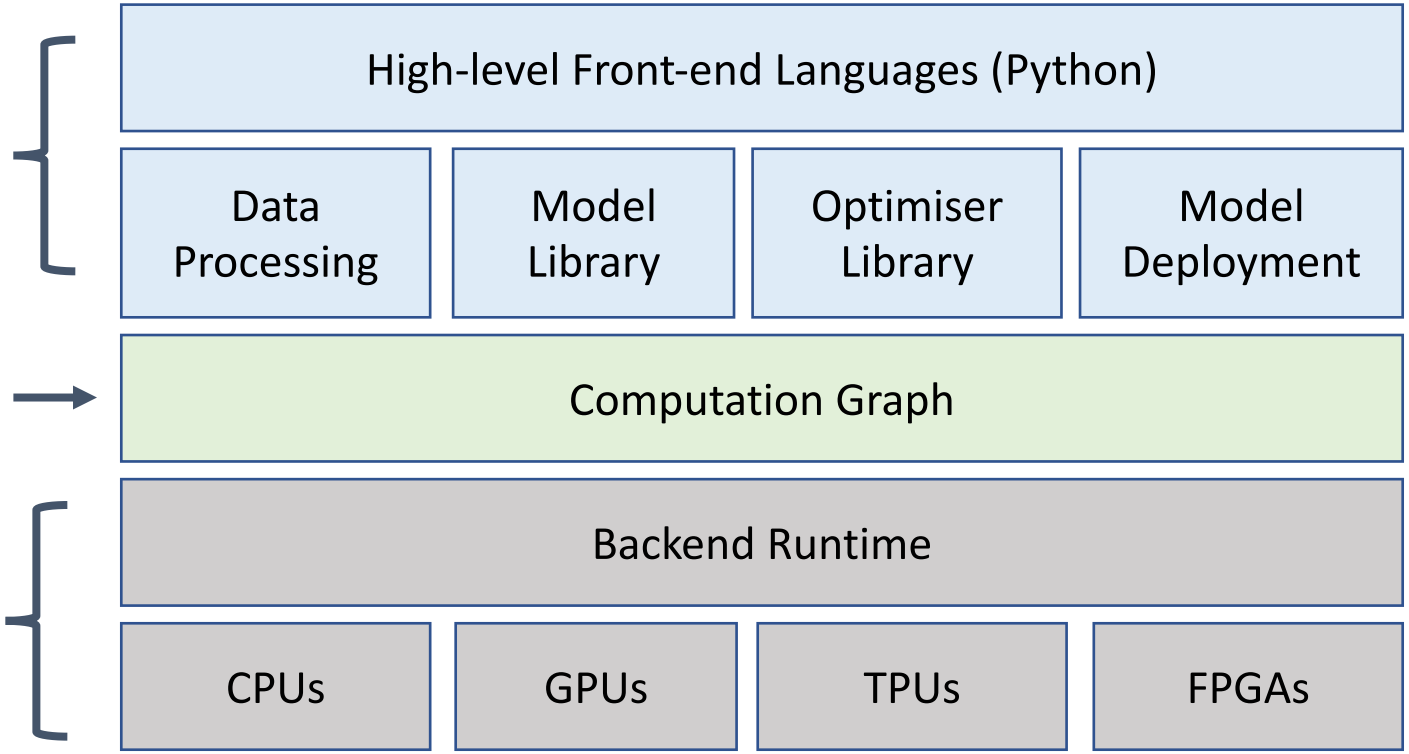
Benefits

- **Simple and flexible** frontend
- **Full life-cycle support**

- **Unified expression** of computation
- **Automatic differentiation**
- Enabling **backend execution**: parallelism, offloading, …

- kernel dispatchers
- Supporting different processors

ML Systems Architecture



Reading

- Optional reading
  - [Deep learning with PyTorch in 60 minutes]
  - [TensorFlow white paper]
  - [PyTorch white paper]

Questions?