esade

Advanced Programming in Python

LECTURE 4

Do Good. Do Better.

Previously On...

Previously On...

Licensing

FOSS

Unittest

Exceptions

Docs

Asserts

Argparse

Git

Introduction

Outline

Previously on...

Introduction

Performance Computing (Overview)

Multithreading

Big O notation

Performance Libraries

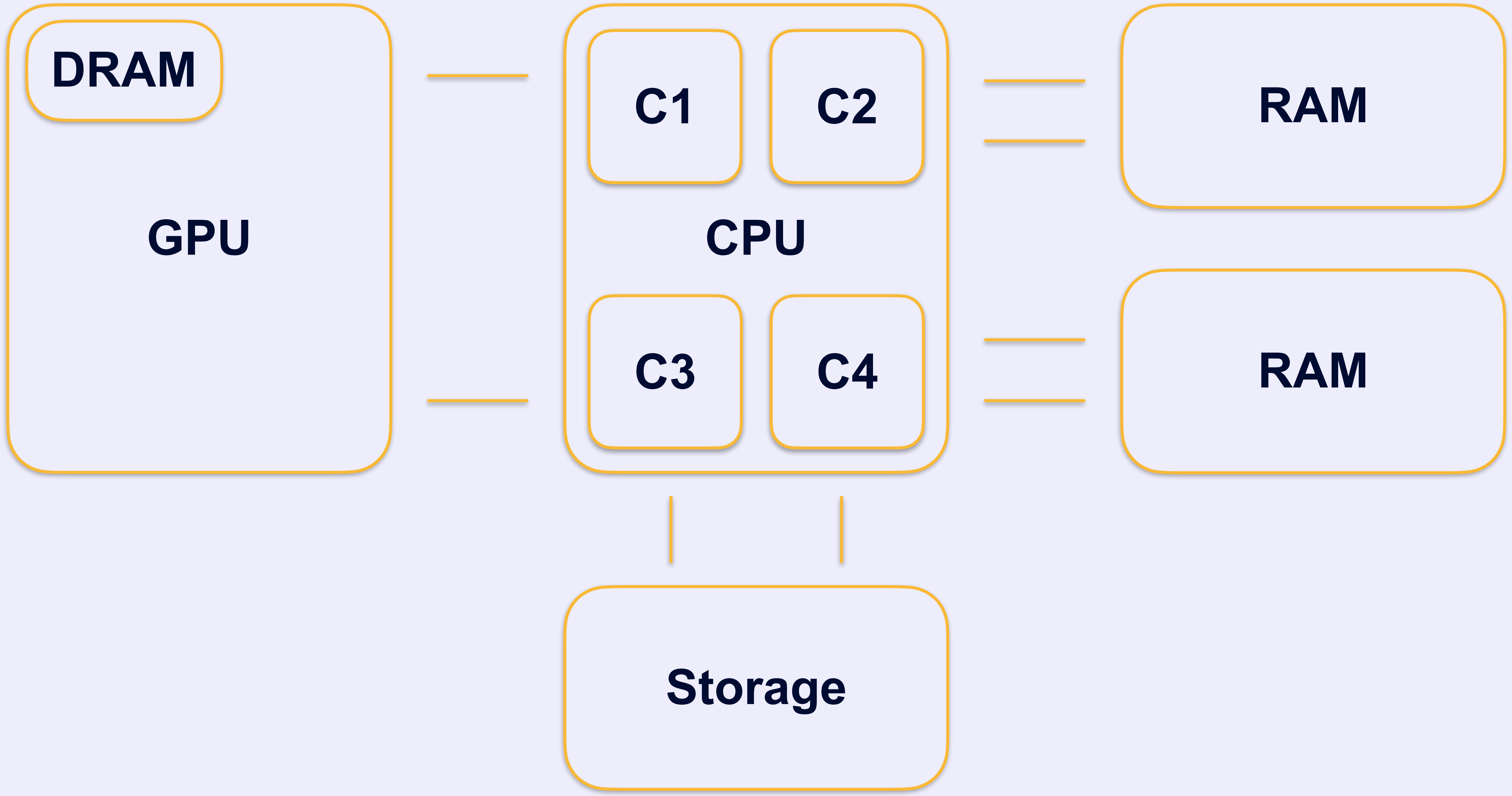
Performance Computing

Overview

Do Good. Do Better.

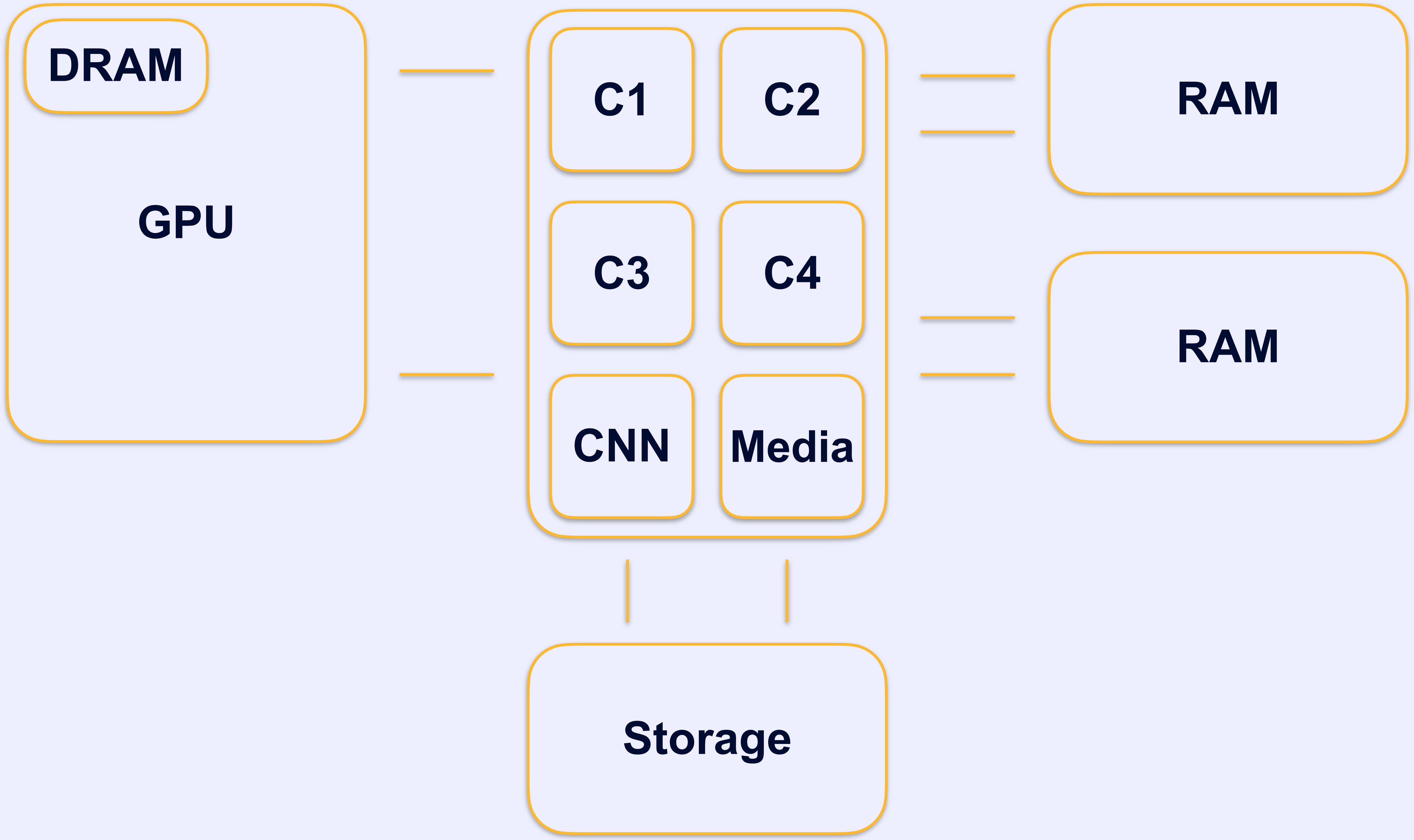
Performance Computing

Hardware: All-purpose



Performance Computing

Hardware: Dedicated Engines



Apple M1 Max Specs, May 2022, https://www.apple.com/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/ (Accessed: 20-05-2022)

Performance Computing

Software

There are “three” different ways that a program can be optimized:

- Instruction optimization (instruccions per cycle)
- Parallelization of code.
- Algorithm optimization.

Multithreading

Multithreading

Parallelism

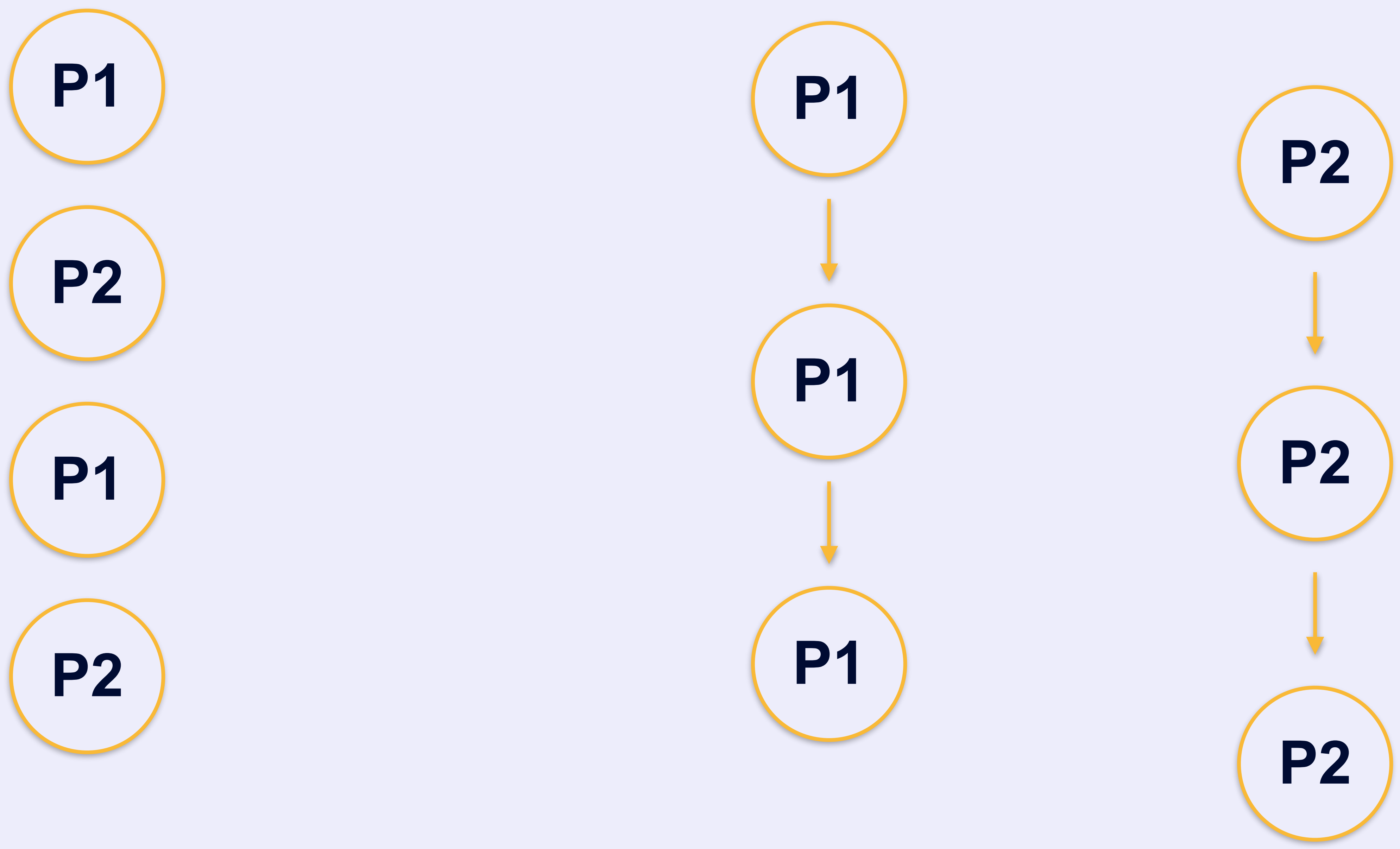
- Allows for an execution of multiple parts of the code in "parallel".
- Allows to perform multiple operations "simultaneously".
- Parallel only means simultaneous if the code is executed in different cores.

Single Core

Thread 1

Multi Core

Process 1 Process 2



Single Core: Threading

- The threading library in python allows to create multiple threads inside the same process.
- The threads are executed in “parallel” in the same core, which means that memory is shared, and a strategy to share variables is needed (semaphores, monitors).
- The threading library is not suitable for parallel computation, but useful for running multiple “activities” or pieces of code at the same time.

Threading python.org, May 2022, https://docs.python.org/3/library/threading.html (Accessed: 20-05-2022)

Single Core: Threading

```
from threading import Thread

class example(Thread):
    def __init__(self):
        super(example, self).__init__()

    def run(self):
        print("this message is from a thread"

if __name__ == "__main__":
    ex = example()
    ex.start()
    print("message from main")
    ex.join(1)
```

Multi Core: Multiprocess

- The multiprocess library in python allows to create multiple process that can live in different CPU cores (real parallelism).
- As the processes are independent, no memory is shared between them (memory has to be copied back and forth).
- The multiprocess library has the class “Pool” that can be used for parallel computation.

Multiprocess python.org, May 2022, https://docs.python.org/3/library/multiprocessing.html (Accessed: 20-05-2022)

Multi Core: Multiprocess

```
from multiprocessing import Pool

def function(arg1):
    return arg1*2

if __name__ == "__main__":
    pl = Pool(2)
    var = [1 2 3 4 5]
    var2 = pl.map(function, var)
    pl.close()
    print(var2)
```

```
from multiprocessing import Process

class example(Process):
    def __init__(self):
        super(example, self).__init__()

    def run(self):
        print("this message is \
            from a process"

if __name__ == "__main__":
    ex = example()
    ex.start()
    print("message from main")
    ex.join(1)
```

Big O notation

Big O: Concept overview

- O notation is used to represent how the complexity/cost of a certain algorithm is going to scale in relation with the amount of data provided to the algorithm.
- It allows to represent “mathematically” the complexity of an algorithm.
- It represents asympthotic behaviour.

Wikipedia O notation, May 2022, https://en.wikipedia.org/wiki/Big_O_notation (Accessed: 9-05-2022)

Big O: Concept overview

- Complexity can be represented in general terms as depending on the number of samples/steps provided to the algorithm (n).
- The most common examples of O notation are the following:

$O(1)$ - Constant

$O(\log n)$ - Logarithmic

$O(n)$ - Linear

$O(n^2)$ - Quadratic

$O(2^n)$ - Expontial

$O(n!)$ - Factorial

Lecture in O notation in python, https://www.youtube.com/watch?v=Ee0HzlnIYWQ (Accessed: 9-05-2022)

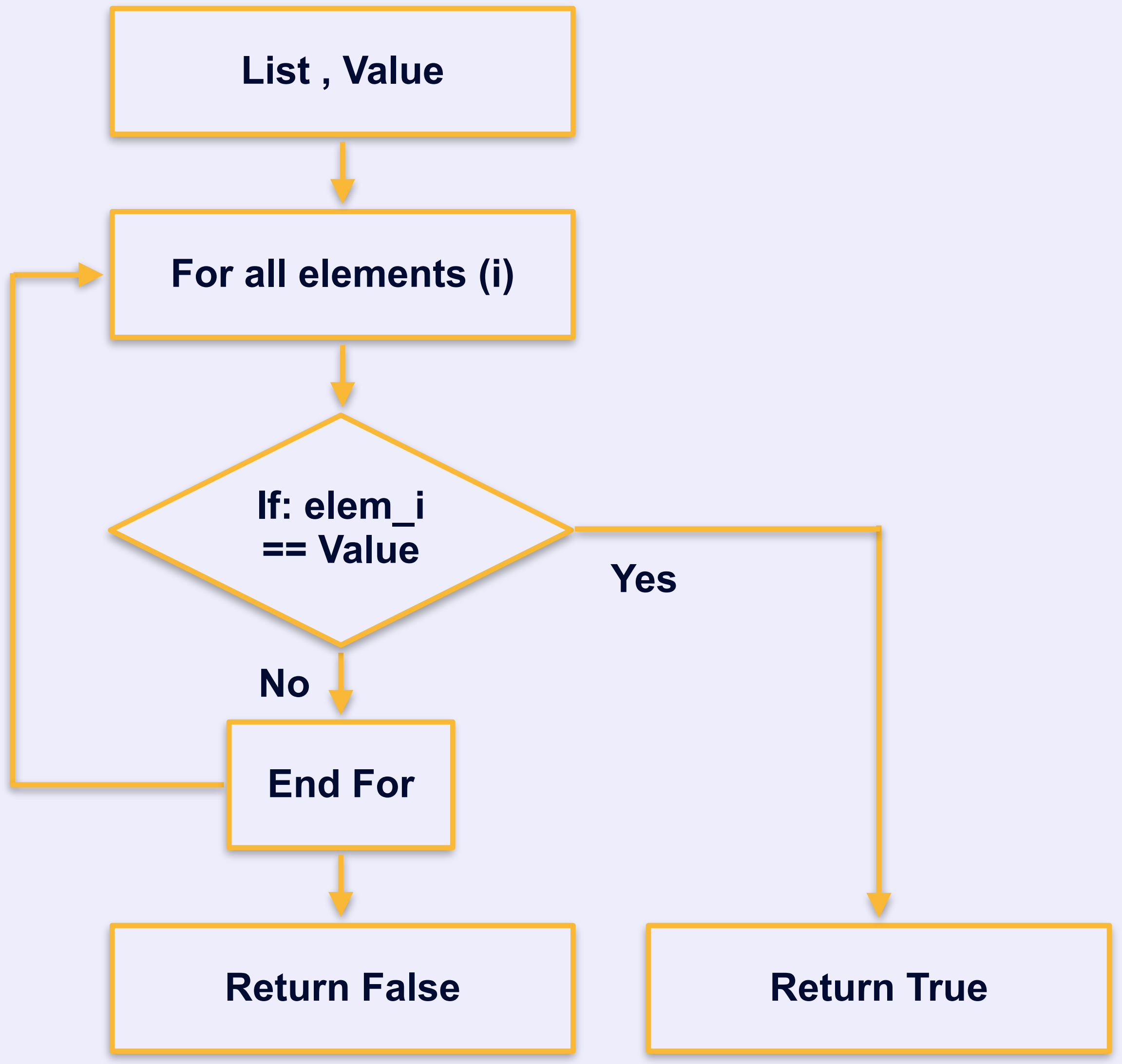
Big O: Constant complex. operation O(1)

```
def function(arg1: int) -> int:
    return arg1 + 1
```

```
def function(arg1: int) -> float:
    var1 = math.sqrt(arg1)
    var2 = var1 + 2*arg1
    return var2
```

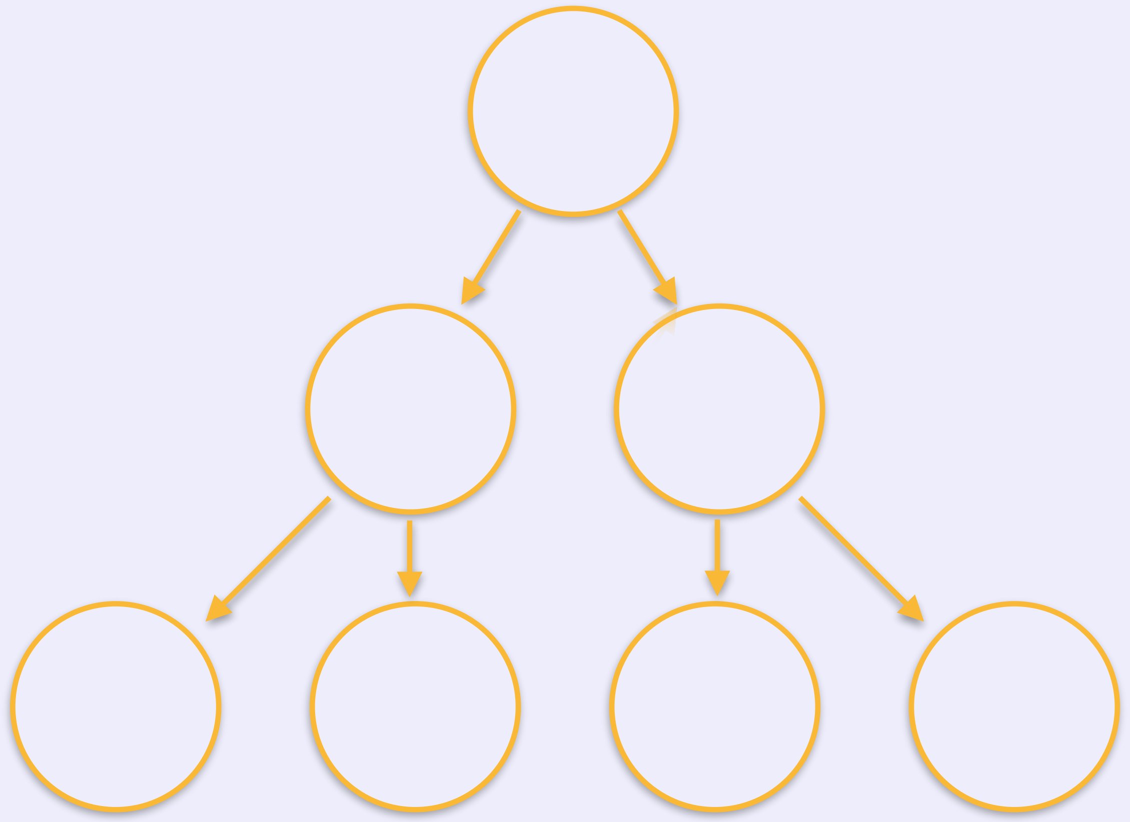
Big O: Naive search O(n)

```python
# naive search (iterate through all elements)
def naive_search(input_list: list, value: int) -> bool:
    for elem in input_list:
        if elem == value:
            return True
    return False
```



Big O: Binary search O(logn)

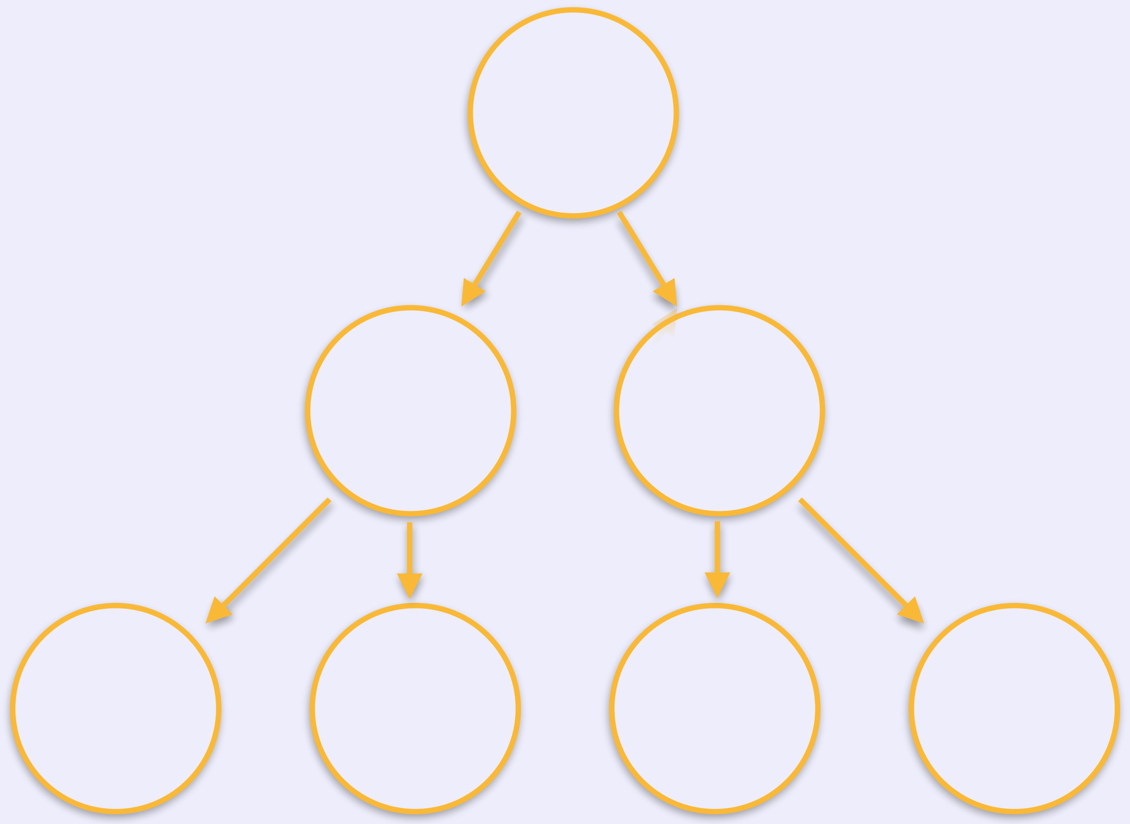
```python
# Binary search
def binary_search(input_list: list, value: int) -> bool:
    l_list = len(input_list)-1
    c_index = int(len(input_list)/2.0)
    while c_index != 0 and c_index != l_list:
        if input_list[c_index] == value:
            return True
        if value > input_list[c_index]:
            c_index = l_list - int((l_list-c_index)/2)
        else:
            c_index = int((c_index)/2)
    return False
```



Wikipedia Binary Search, April 2022, https://en.wikipedia.org/wiki/Binary_search_algorithm (Accessed: 20-04-2022)

Big O: Fibonacci O(2^n)

```
def rec_fibo_student(current_num: int) -> int:
    if current_num <= 1 :
        return current_num
    else:
        return rec_fibo_student(current_num-1) + \
            rec_fibo_student(current_num-2)
```



esade

Performance Libraries

Do Good. Do Better.

Performance libraries: numpy

- Numpy is a scientific library for numerical analysis and computation.
- It allows us to handle arrays and matrices of data.
- It provides methods for different algebraic operations while abstracting its underlying implementation.
- Implements the creation subsets of numbers, such as random numbers or even linear/logarithmic spaces.

Performance libraries: pytorch

- Pytorch is primarely a deep learning library which allows to develop/train/execute Deep CNNs.
- Pytorch allows to seamlessly use matrix operations with underlying parallelization both in CPU and GPU.
- Provides most of the functionality of numpy, with methods to exchange data types.
- Main base type is called tensor.

esade

Do Good. Do Better.