Introduction to Parallelism in Python

CodeSeoul MLA
December 9, 2023

Overview

- Basics
- Multiprocessing API
- Sharing memory
- Further interest
- Application in ML

Why?

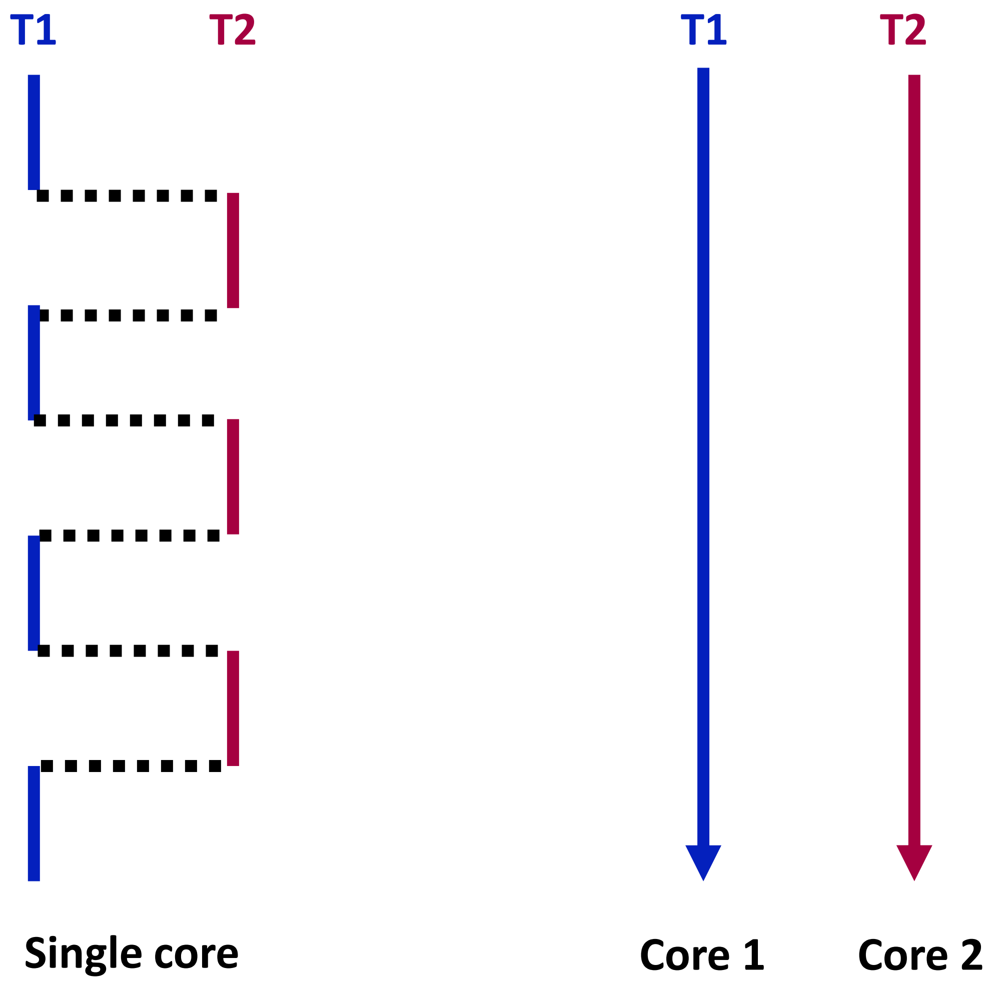
- Increased performance
- Efficient resource utilization
- Scalability
- Complex problem solving

Concurrent

- Tasks make progress independently, but not necessarily simultaneously.

Parallel

- Tasks are executed simultaneously on multiple processors or cores.

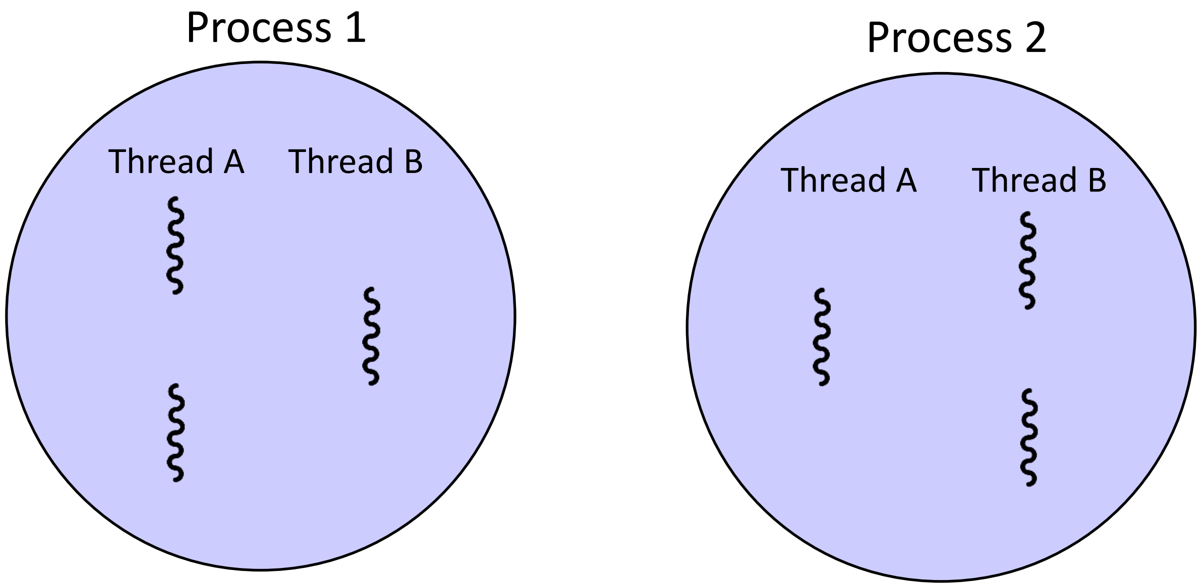


Thread

- Shares memory space with other threads in the same process
- Communicates directly through shared memory
- Lower overhead

Process

- Independent program with its own memory space
- Interprocess communication (IPC) is required for communication
- Higher overhead



Example in Python

```
Thread(target=func, args=(args,)).start()
```

```
Process(target=func, args=(args,)).start()
```

_thread.py

```python
import threading

# Define a simple function that will be executed in a thread
def print_numbers():
    for i in range(5):
        print(i)

if __name__ == "__main__":
    # Create a Thread object and target it to the function
    my_thread = threading.Thread(target=print_numbers)

    # Start the thread
    my_thread.start()
    # Wait for the thread to finish (optional)
    my_thread.join()

    # Continue with the main program
    print("Main program continues...")
```

_process.py

```python
import multiprocessing

# Define a simple function that will be executed in a process
def print_numbers():
    for i in range(5):
        print(i)

if __name__ == "__main__":
    # Create a Process object and target it to the function
    my_process = multiprocessing.Process(target=print_numbers)

    # Start the process
    my_process.start()
    # Wait for the process to finish (optional)
    my_process.join()

    # Continue with the main program
    print("Main program continues...")
```

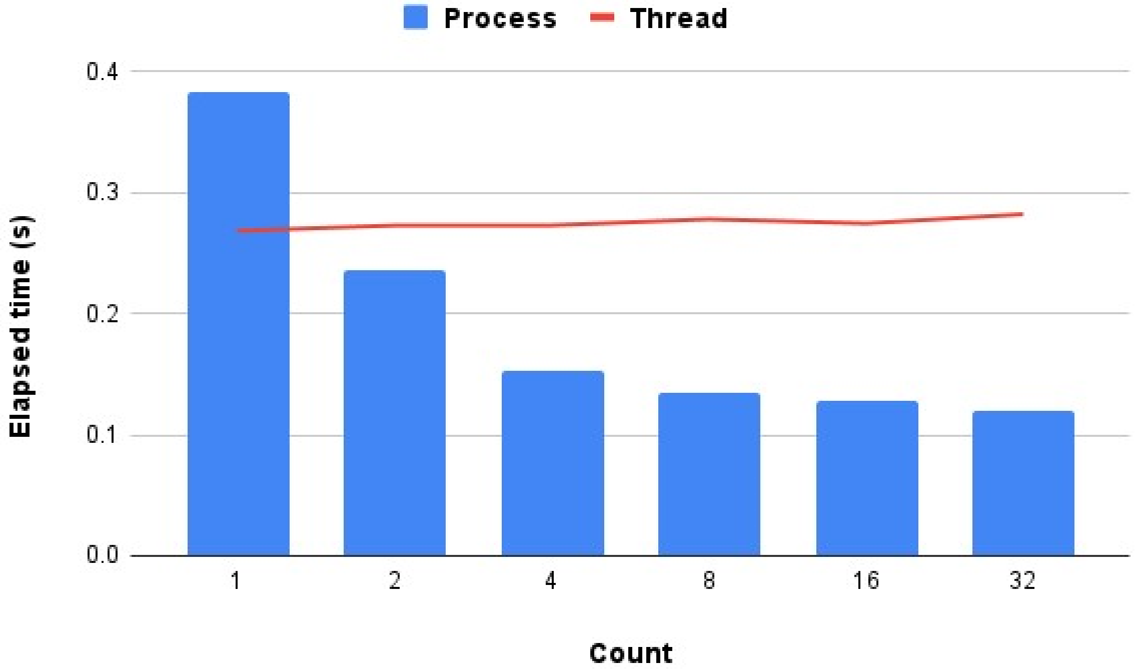
Thread vs Process

Let's experiment on embarrassingly parallel task:

Task is to get square for each element of

`list(range(1, 1000001))`

- Using thread
- Using process
- Varying number of each



Increasing the number of processes leads to quicker results **but increasing the number of threads does not** provide similar benefits.

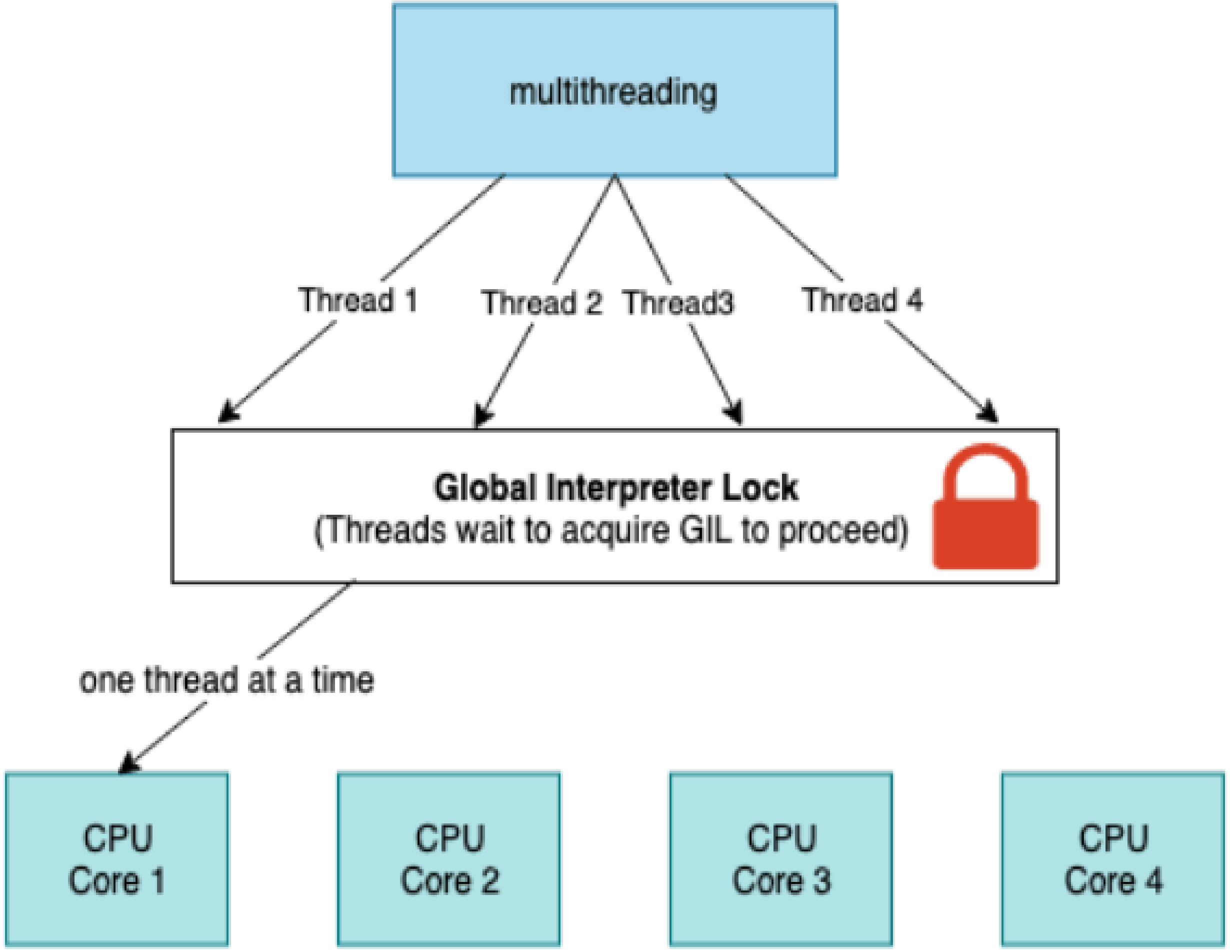
Any ideas, why?

Global Interpreter Lock

Global Interpreter Lock

In Python, the **interpreter** permits the **execution** of **only one thread at any given moment**.

The **Global Interpreter Lock** (**GIL**) is responsible for imposing this limitation.

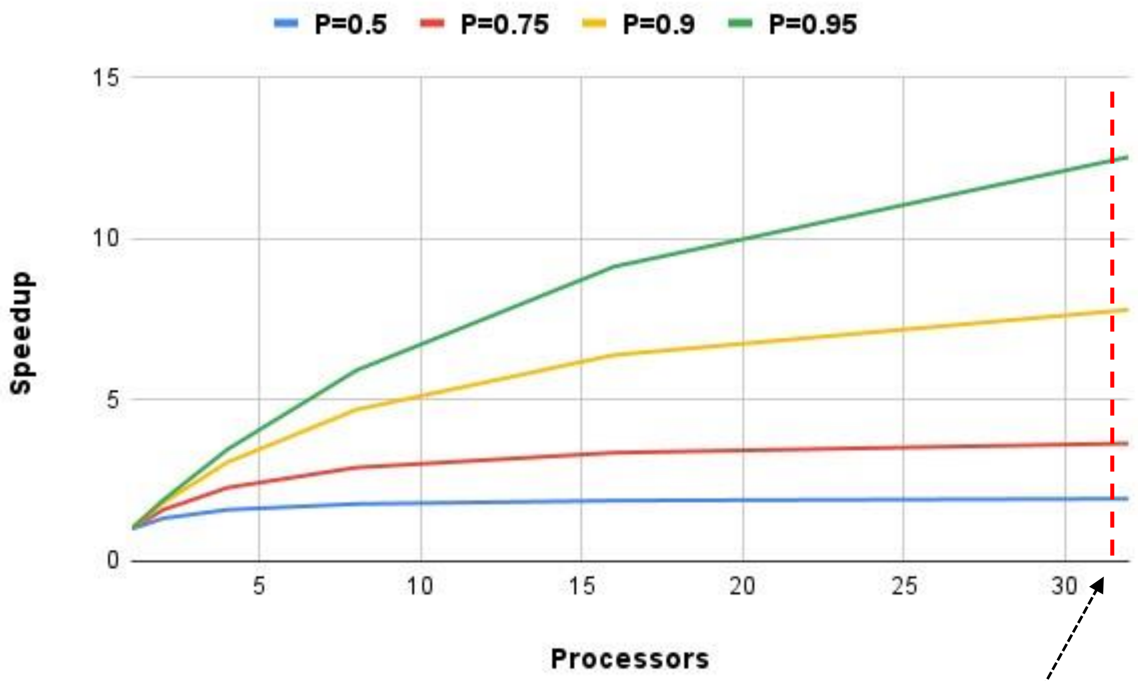


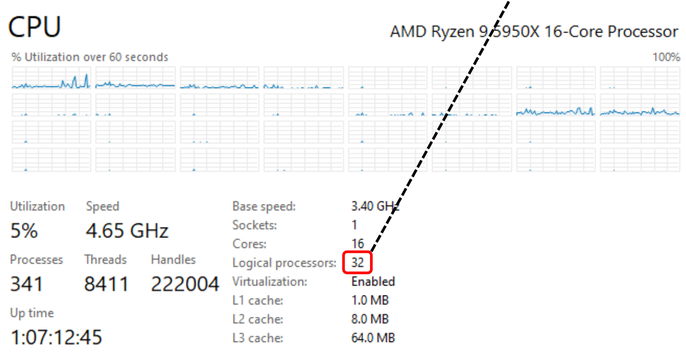
Amdahl's Law

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}$$

$\boldsymbol{S(n)}$: theoretical speedup
$\boldsymbol{P}$: fraction of the algorithm that can be made parallel
$\boldsymbol{n}$: number of CPU threads

- The ***amount of speedup*** a program will see by using $\boldsymbol{n}$ processors is based on ***how much*** of the program is ***parallel*** (can be split up among multiple CPU cores).





CPU
AMD Ryzen 9 5950X 16-Core Processor

Utilization
5%

Speed
4.65 GHz

Processes
341

Threads
8411

Handles
222004

Up time
1:07:12:45

Base speed: 3.40 GHz
Sockets: 1
Cores: 16
Logical processors: 32
Virtualization: Enabled
L1 cache: 1.0 MB
L2 cache: 8.0 MB
L3 cache: 64.0 MB

From "taskmgr.exe"

Starting Python process

_tracker.py

```python
import os, time, multiprocessing

def worker1():
    while True:
        time.sleep(2)
        print(f"Process PID {os.getpid()} is running")

def worker2():
    while True:
        time.sleep(2)
        print(f"Process PID {os.getpid()} is running")

if __name__ == "__main__":
    print("ID of main process: {}".format(os.getpid()))

    # creating processes
    p1 = multiprocessing.Process(target=worker1)
    p2 = multiprocessing.Process(target=worker2)

    # starting processes
    p1.start()
    p2.start()

    # process IDs
    print(f"Process PID for worker1: {p1.pid}")
    print(f"Process PID for worker1: {p2.pid}")
```

This is a task running in a separate process

This is also a task running in a separate process

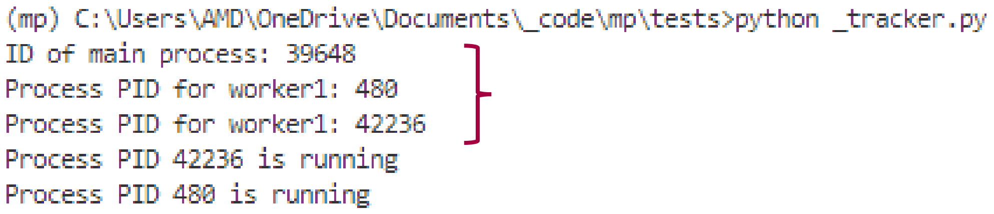
Two processes that we create here are inside a process already (inside main process)

Let’s check if these processes did really start

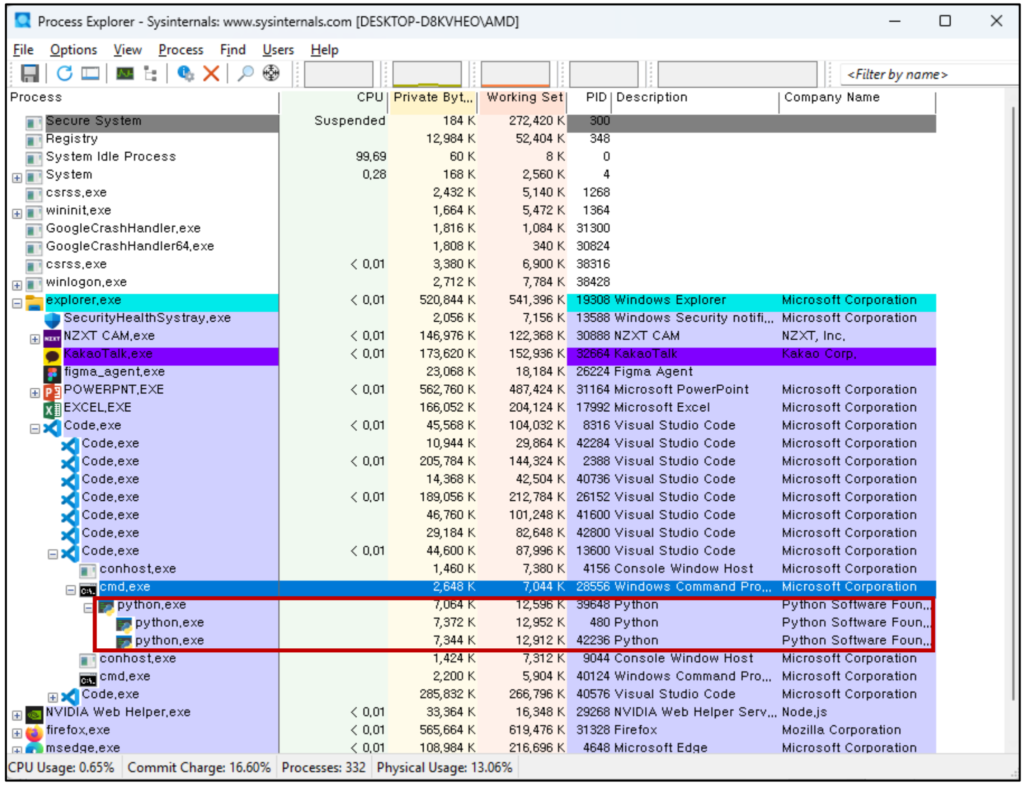
Starting Python process

- In **Windows**, we can download Process Explorer:

from https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer



```
(mp) C:\Users\AMD\OneDrive\Documents\_code\mp\tests>python _tracker.py
ID of main process: 39648
Process PID for worker1: 480
Process PID for worker1: 42236
Process PID 42236 is running
Process PID 480 is running
```

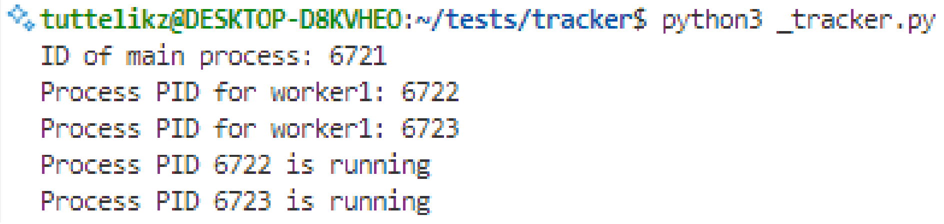


Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-D8KVHEO\AMD]

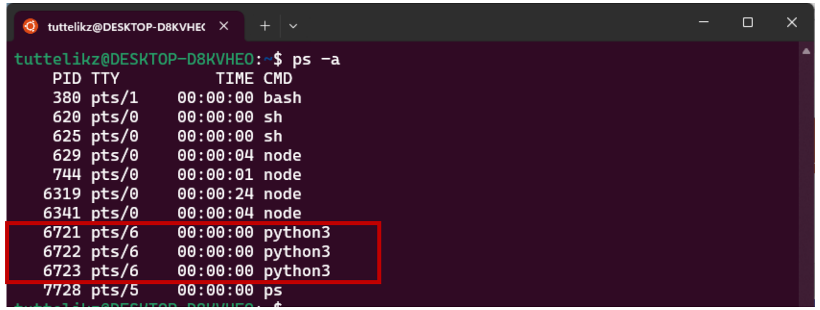
Process	CPU	Private Byt...	Working Set	PID	Description	Company Name
Secure System	Suspended	184 K	272,420 K	300		
Registry		12,984 K	52,404 K	348		
System Idle Process	99,69	60 K	8 K	0		
System	0,28	168 K	2,560 K	4		
csrss,exe		2,432 K	5,140 K	1268		
wininit,exe		1,664 K	5,472 K	1364		
GoogleCrashHandler,exe		1,816 K	1,084 K	31300		
GoogleCrashHandler64,exe		1,808 K	340 K	30824		
csrss,exe	< 0,01	3,380 K	6,900 K	38316		
winlogon,exe		2,712 K	7,784 K	38428		
explorer,exe	< 0,01	520,844 K	541,396 K	19308	Windows Explorer	Microsoft Corporation
SecurityHealthSystray,exe		2,056 K	7,156 K	13588	Windows Security notifi...	Microsoft Corporation
NZXT CAM,exe	< 0,01	146,976 K	122,368 K	30888	NZXT CAM	NZXT, Inc,
KakaoTalk,exe	< 0,01	173,620 K	152,936 K	32664	KakaoTalk	Kakao Corp,
figma_agent,exe		23,068 K	18,184 K	26224	Figma Agent	
POWERPNT,EXE	< 0,01	562,760 K	487,424 K	31164	Microsoft PowerPoint	Microsoft Corporation
EXCEL,EXE		166,052 K	204,124 K	17992	Microsoft Excel	Microsoft Corporation
Code,exe	< 0,01	45,568 K	104,032 K	8316	Visual Studio Code	Microsoft Corporation
Code,exe		10,944 K	29,864 K	42284	Visual Studio Code	Microsoft Corporation
Code,exe	< 0,01	205,784 K	144,324 K	2388	Visual Studio Code	Microsoft Corporation
Code,exe		14,368 K	42,504 K	40736	Visual Studio Code	Microsoft Corporation
Code,exe	< 0,01	189,056 K	212,784 K	26152	Visual Studio Code	Microsoft Corporation
Code,exe		46,760 K	101,248 K	41600	Visual Studio Code	Microsoft Corporation
Code,exe		29,184 K	82,648 K	42800	Visual Studio Code	Microsoft Corporation
Code,exe	< 0,01	44,600 K	87,996 K	13600	Visual Studio Code	Microsoft Corporation
conhost,exe		1,460 K	7,380 K	4156	Console Window Host	Microsoft Corporation
cmd,exe		2,648 K	7,044 K	28556	Windows Command Pro...	Microsoft Corporation
python,exe		7,064 K	12,596 K	39648	Python	Python Software Foun...
python,exe		7,372 K	12,952 K	480	Python	Python Software Foun...
python,exe		7,344 K	12,912 K	42236	Python	Python Software Foun...
conhost,exe		1,424 K	7,312 K	9044	Console Window Host	Microsoft Corporation
cmd,exe		2,200 K	5,904 K	40124	Windows Command Pro...	Microsoft Corporation
Code,exe		285,832 K	266,796 K	40576	Visual Studio Code	Microsoft Corporation
NVIDIA Web Helper,exe	< 0,01	33,364 K	16,348 K	29268	NVIDIA Web Helper Serv...	Node.js
firefox,exe	< 0,01	565,664 K	619,476 K	31328	Firefox	Mozilla Corporation
msedge,exe	< 0,01	108,984 K	216,696 K	4648	Microsoft Edge	Microsoft Corporation

CPU Usage: 0.65% | Commit Charge: 16.60% | Processes: 332 | Physical Usage: 13.06%

- In **Linux**, try “ps -a”

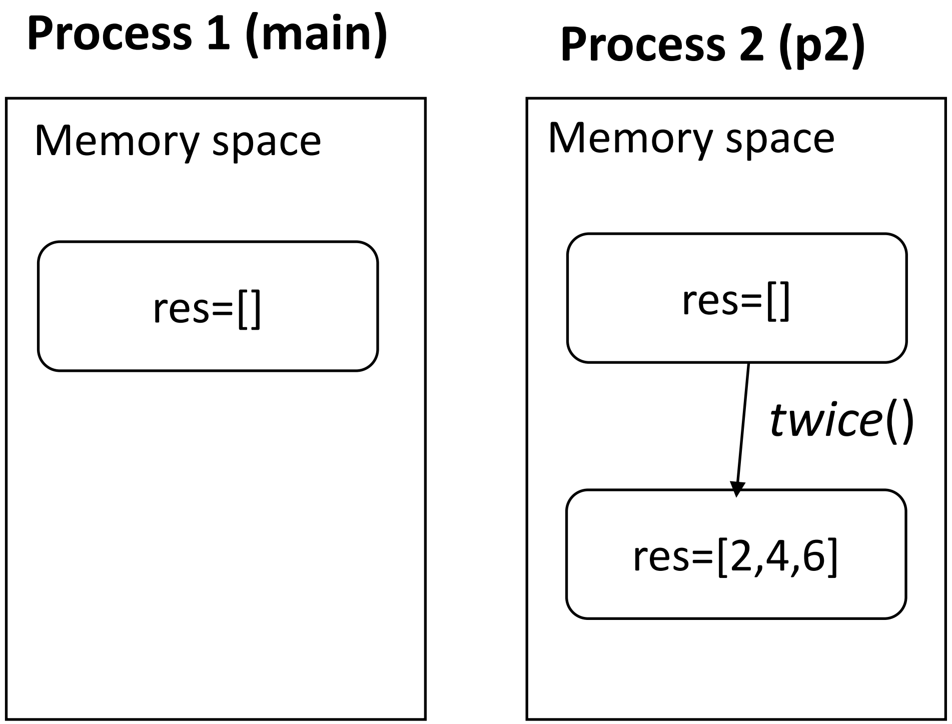


```
tuttelikz@DESKTOP-D8KVHEO:~/tests/tracker$ python3 _tracker.py
  ID of main process: 6721
  Process PID for worker1: 6722
  Process PID for worker1: 6723
  Process PID 6722 is running
  Process PID 6723 is running
```



```
tuttelikz@DESKTOP-D8KVHEO:~$ ps -a
  PID TTY          TIME CMD
  380 pts/1    00:00:00 bash
  620 pts/0    00:00:00 sh
  625 pts/0    00:00:00 sh
  629 pts/0    00:00:04 node
  744 pts/0    00:00:01 node
 6319 pts/0    00:00:24 node
 6341 pts/0    00:00:04 node
 6721 pts/6    00:00:00 python3
 6722 pts/6    00:00:00 python3
 6723 pts/6    00:00:00 python3
 7728 pts/5    00:00:00 ps
```

Sharing memory



Process 1 (main)

Process 2 (p2)

Memory space

res=[]

Memory space

res=[]

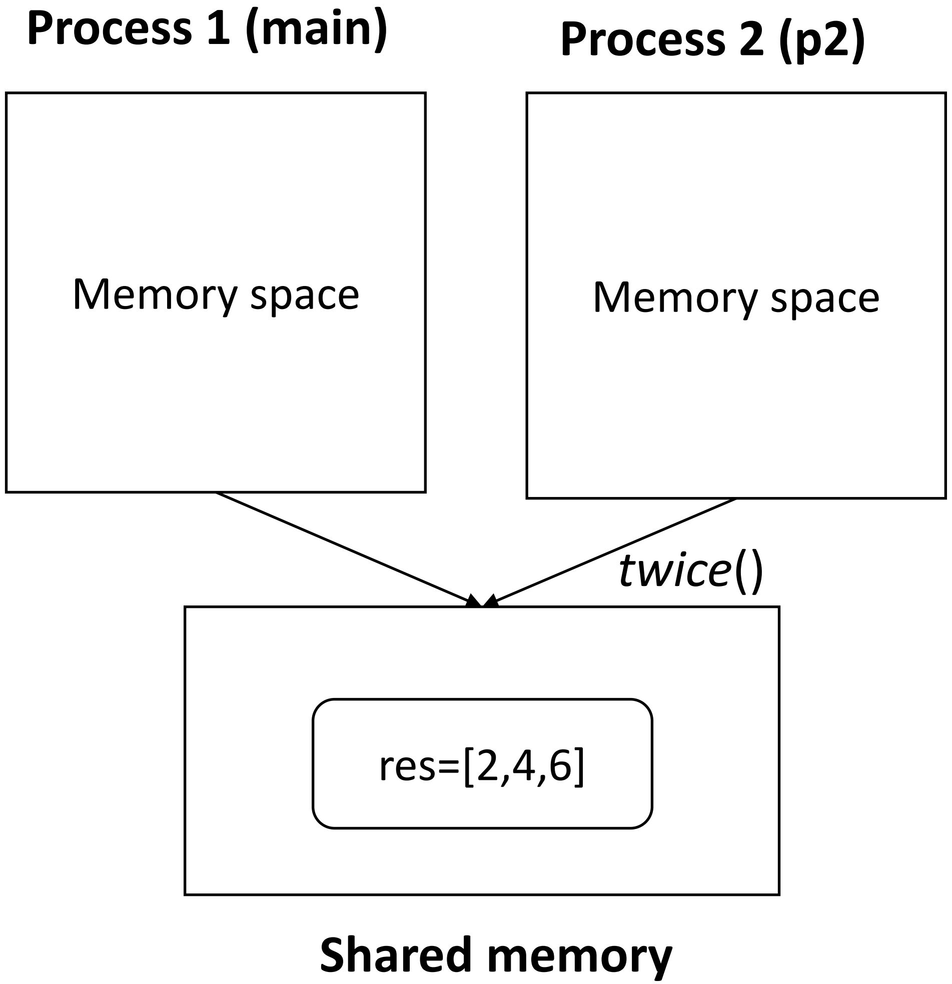
twice()

res=[2,4,6]

_sharing-memory1.py

```
from multiprocessing import Process
res = []

def twice(mylist):
    global res
    for num in mylist:
        res.append(num * 2)
    print(f"res(in process p2): {res}")

if __name__ == "__main__":
    mylist = [1,2,3]
    # creating new process
    p2 = Process(target=twice, args=(mylist,))
    # starting process
    p2.start()
    # wait until process is finished
    p2.join()

    print(f"res(in main program): {res}")
```

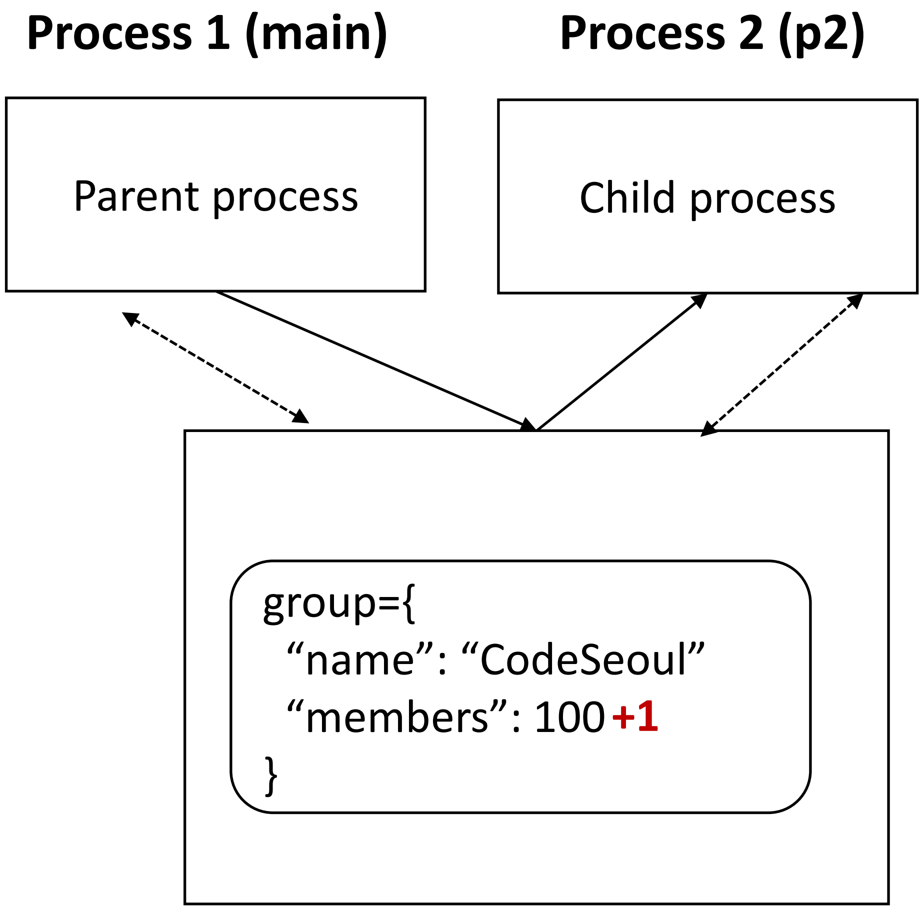
Sharing memory



_sharing-memory2.py

```
import multiprocessing
from multiprocessing import Process, Value, Array

def twice(mylist, res):
    for idx, num in enumerate(mylist):
        res[idx] = num * 2

    print(f"res(in process p2): {res[:]}")

if __name__ == "__main__":
    mylist = [1,2,3]
    # creating Array of int data type with space of 3
    res = multiprocessing.Array('i', 3)
    # creating new process
    p2 = multiprocessing.Process(target=twice,
args=(mylist, res))
    # starting process
    p2.start()
    # wait until the process is finished
    p2.join()

    # print result array
    print(f"Result(in main program): {res[:]}")
```

Manager

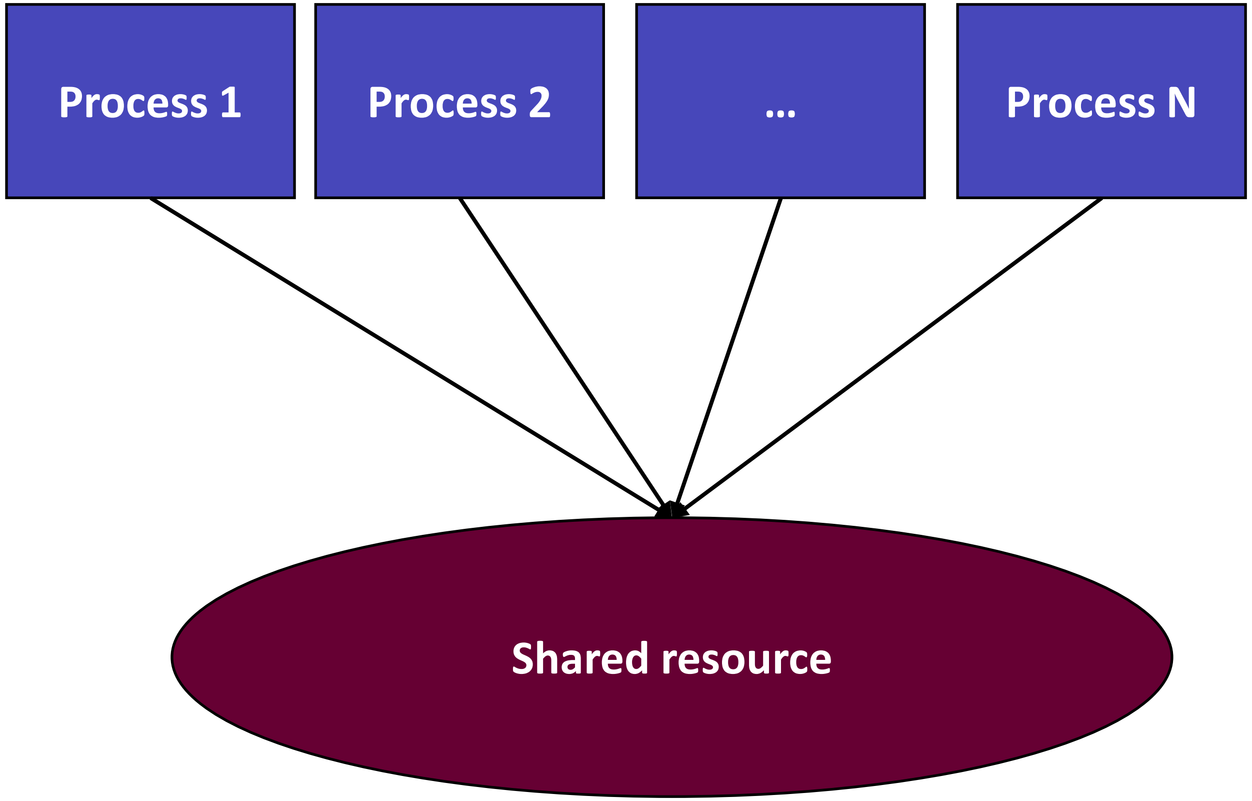


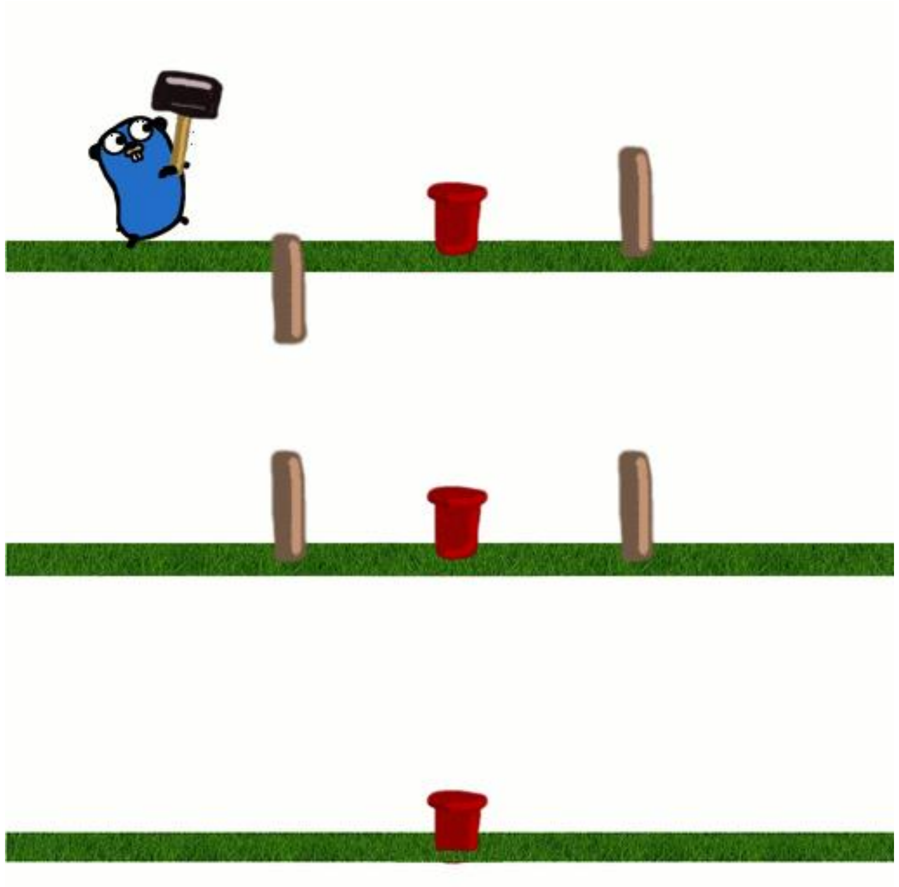
Server process controlled by Manager

_manager.py

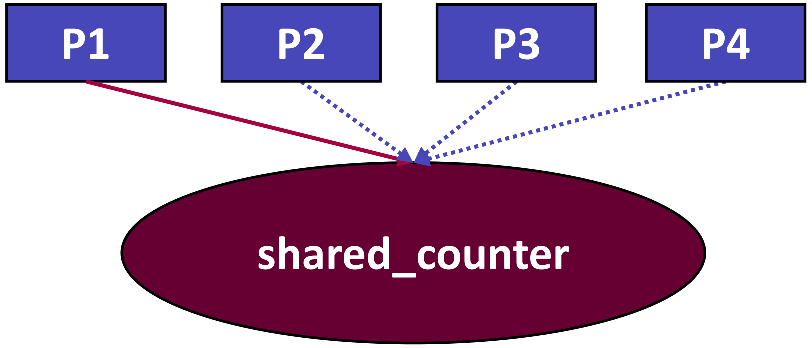
```
from multiprocessing import Process, Manager

def _addmember(d):
    d["members"]+=1
    print(f"Welcome to {d['name']}, we are
{d['members']} now")

if __name__ == '__main__':
    manager = Manager()
    group = manager.dict()
    group["name"] = "CodeSeoul"
    group["members"] = 100
    p2 = Process(target=_addmember,args=(group,))
    p2.start()
    p2.join()

    print(f"Result(in main program):
{group['members']}")
```

Race condition

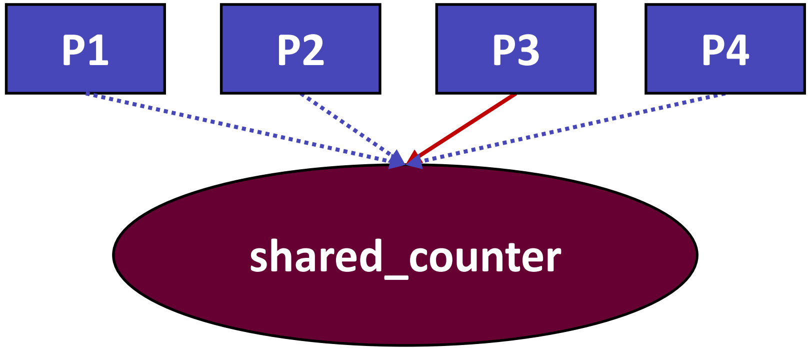




Lock



“**with lock**” ensures that only one process can access and modify the shared counter at a time

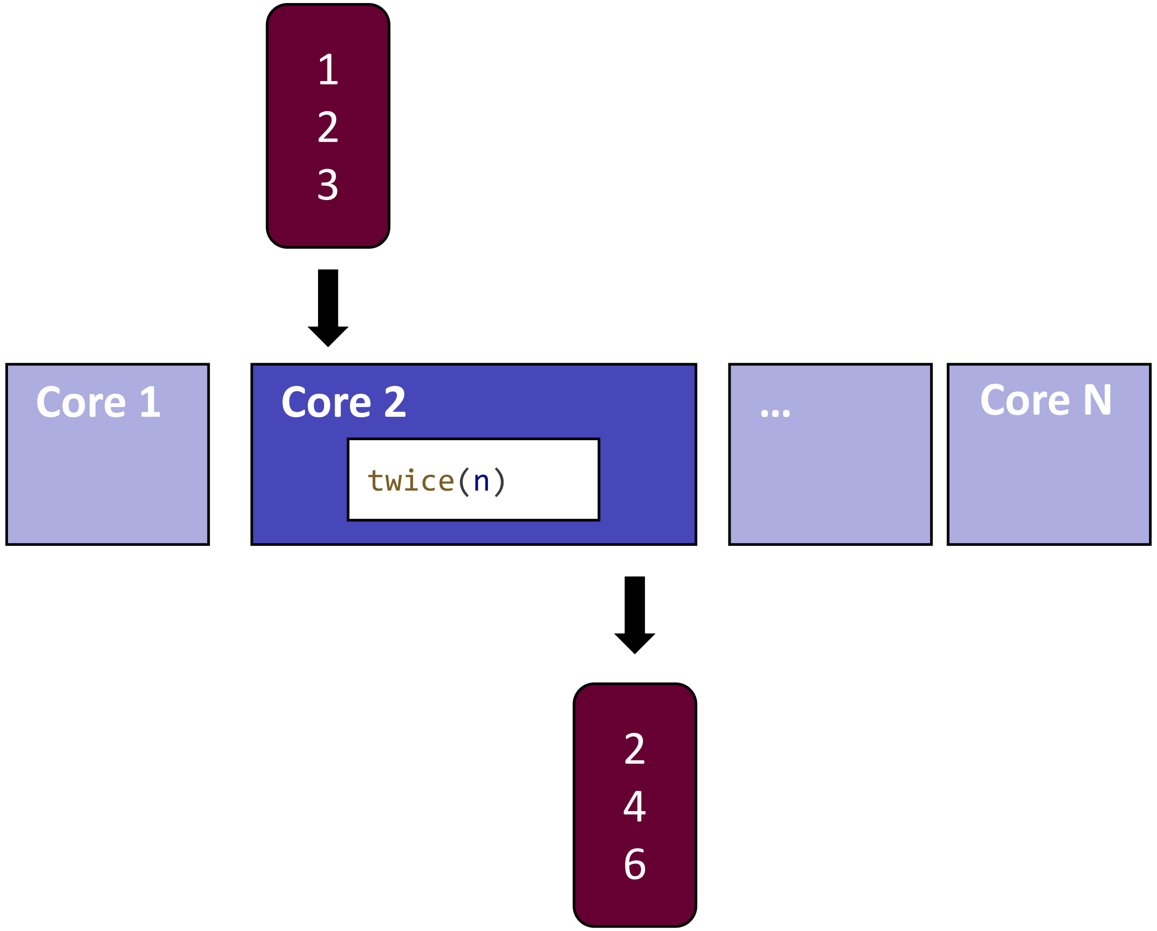


_lock2.py

*Also try to run lock1.py

```
from multiprocessing import Process, Value, Lock

def _increment(shared_counter, lock):
    for _ in range(1000):
        with lock:
            shared_counter.value += 1

if __name__ == '__main__':
    shared_counter = Value('i', 0)
    lock = Lock()
    processes = []
    for _ in range(4):
        process = Process(
            target=_increment, args=(shared_counter,
lock))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    print("Final Counter Value:", shared_counter.value)
```

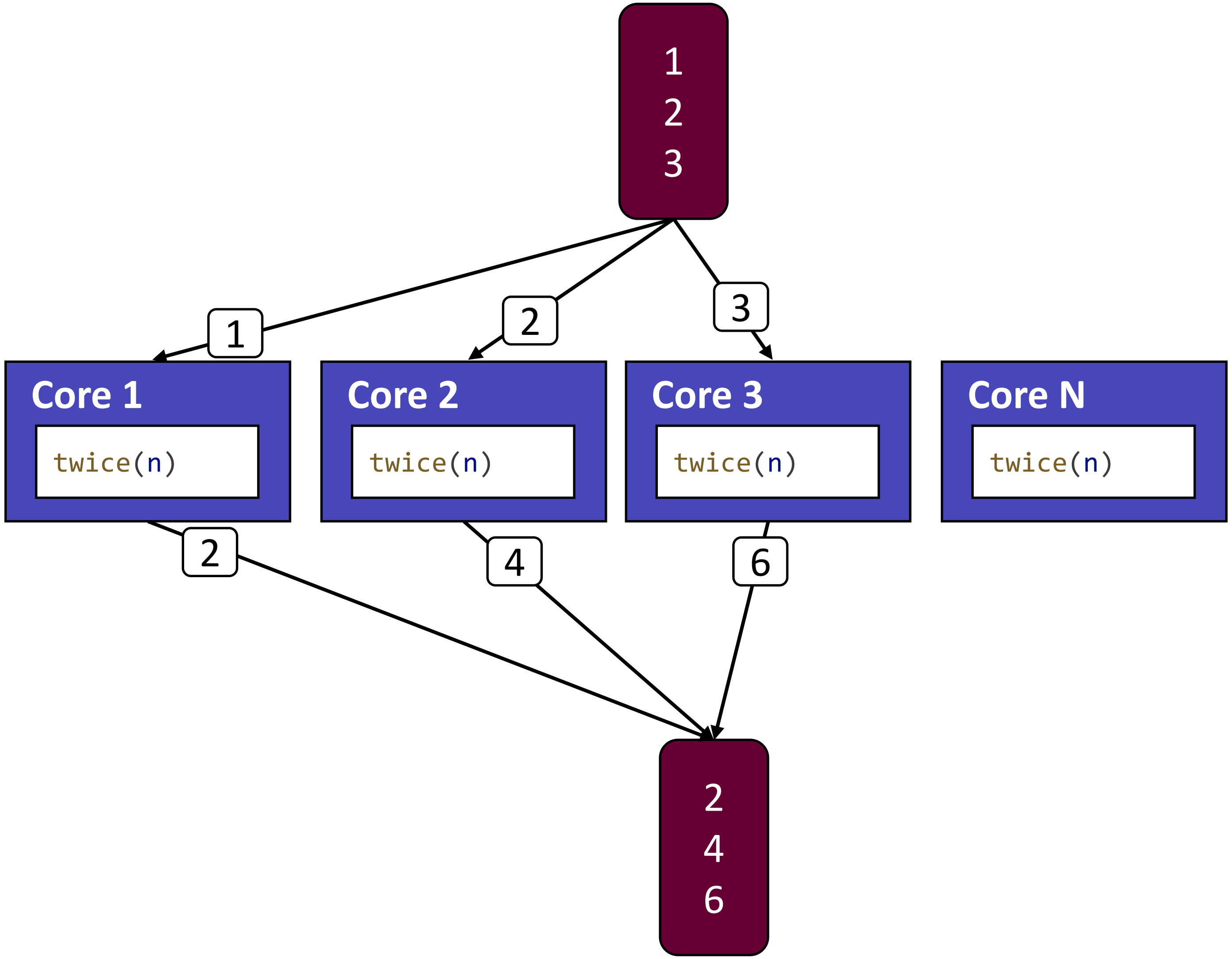
Pool



_pool1.py

```
def twice(n):
    return (n*2)


if __name__ == "__main__":
    mylist = [1, 2, 3]

    res = []
    for num in mylist:
        res.append(twice(num))

    print(res)
```

Pool

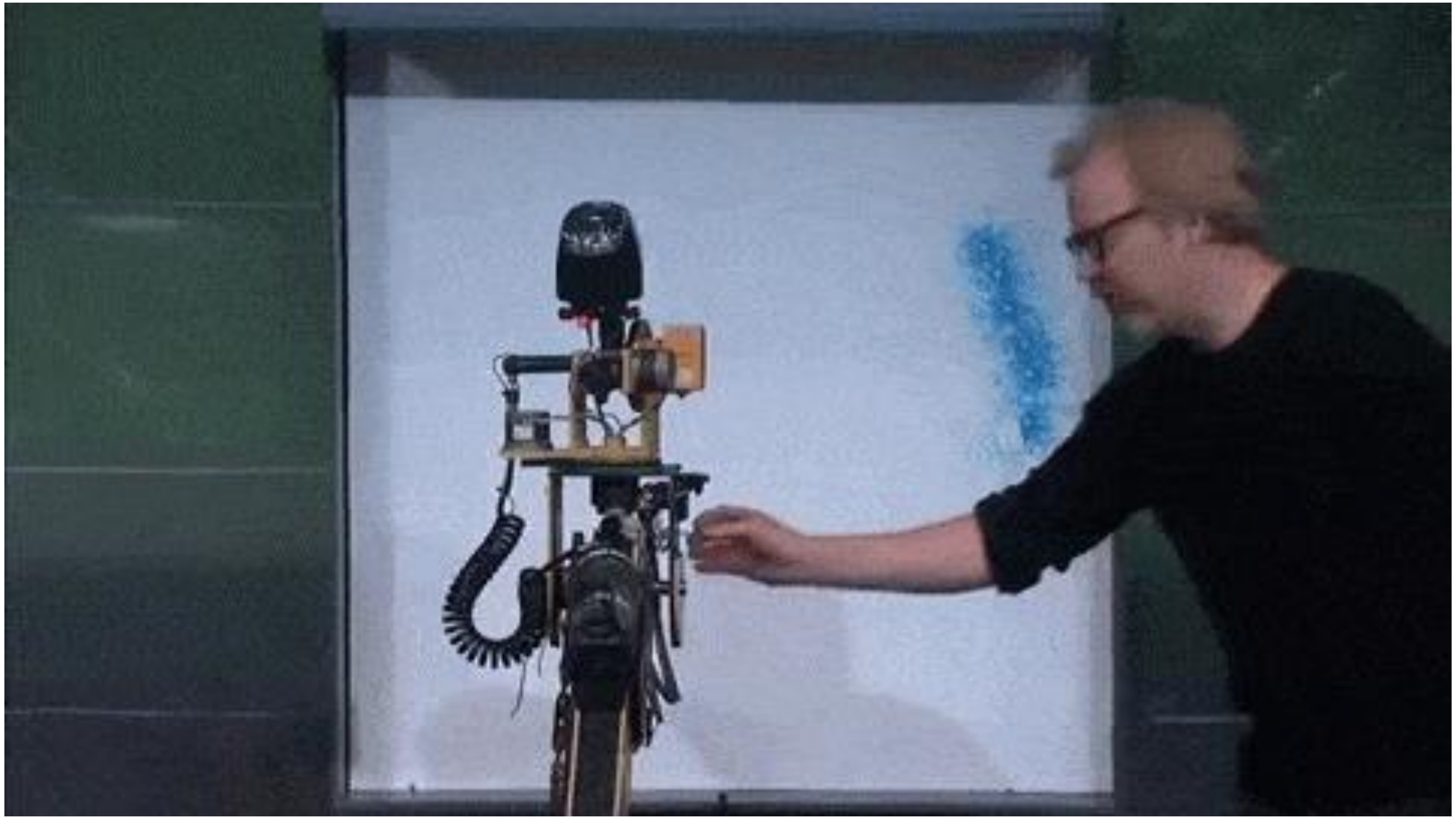


_pool2.py

```
import os
import multiprocessing

def twice(n):
    print(f"PID for {n}: {os.getpid()}")
    return (n*2)

if __name__ == "__main__":
    mylist = [1,2,3]

    p = multiprocessing.Pool()
    res = p.map(twice, mylist)

    print(res)
```

CPU

- **Multiple cores**



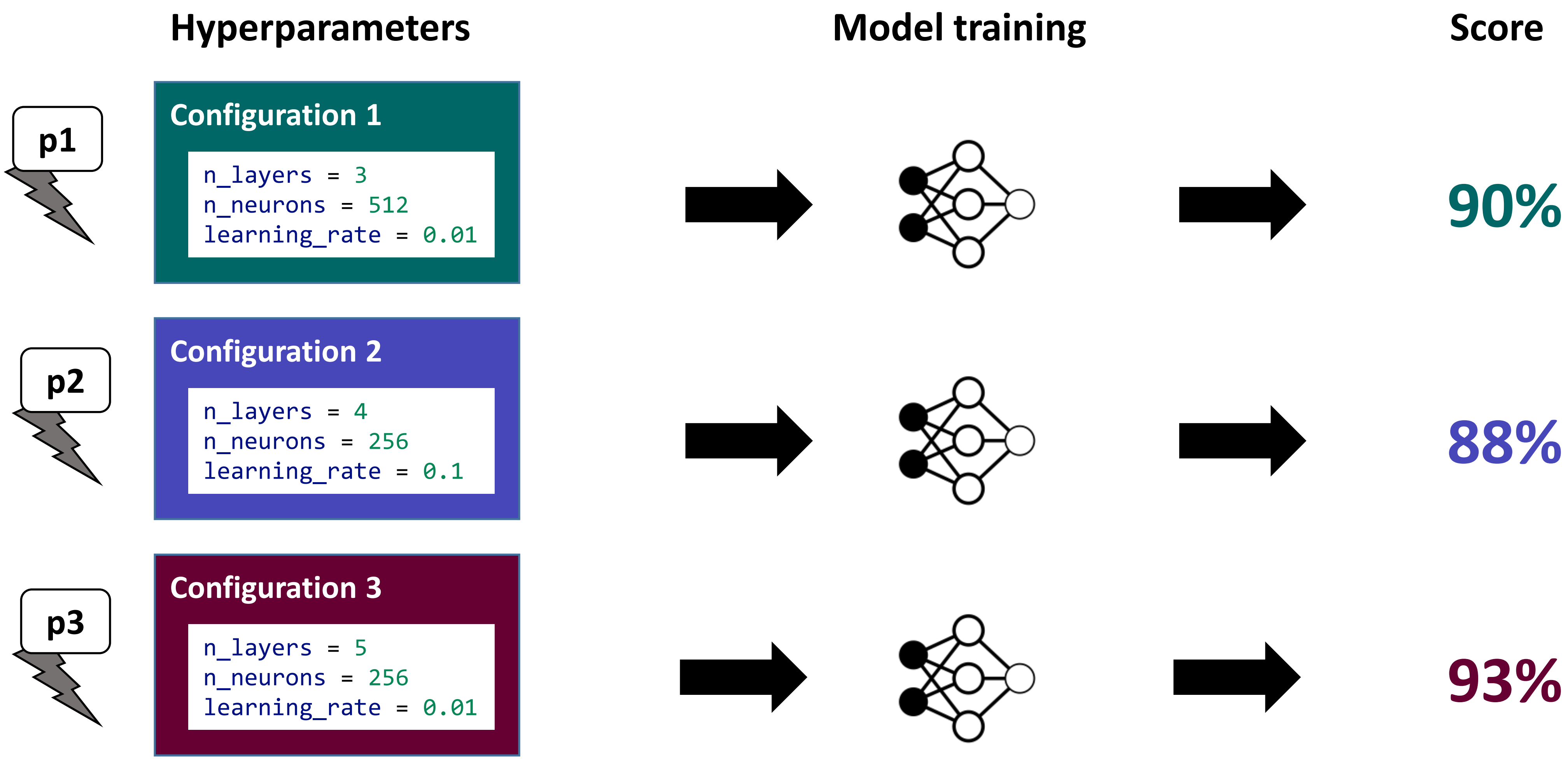
YouTube: Mythbusters Demo GPU versus CPU

GPU

- **Hundreds of cores!**

YouTube: Mythbusters Demo GPU versus CPU

Applications in ML | Hyperparameter optimization



Configuration 1

```
n_layers = 3
n_neurons = 512
learning_rate = 0.01
```

90%

Configuration 2

```
n_layers = 4
n_neurons = 256
learning_rate = 0.1
```

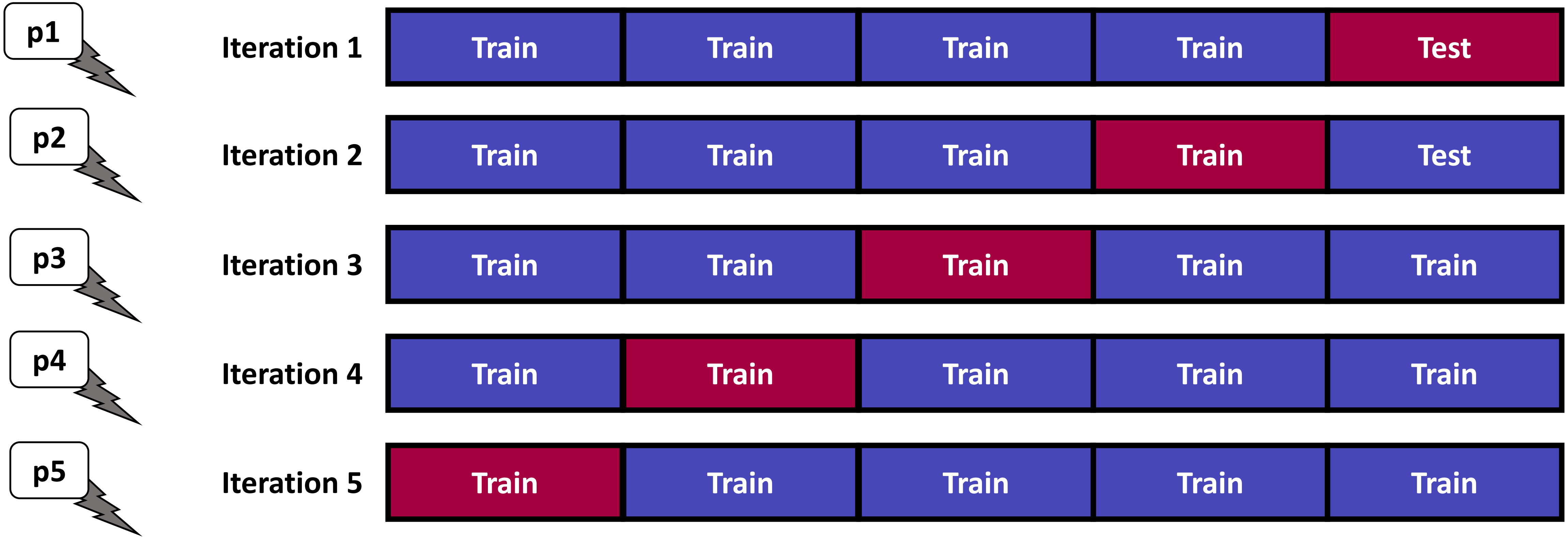
88%

Configuration 3

```
n_layers = 5
n_neurons = 256
learning_rate = 0.01
```

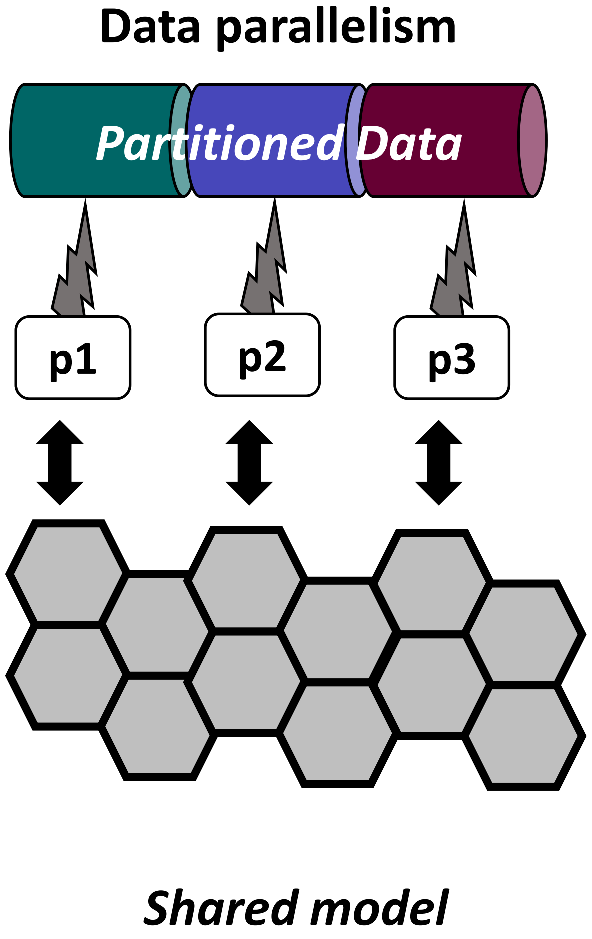
93%

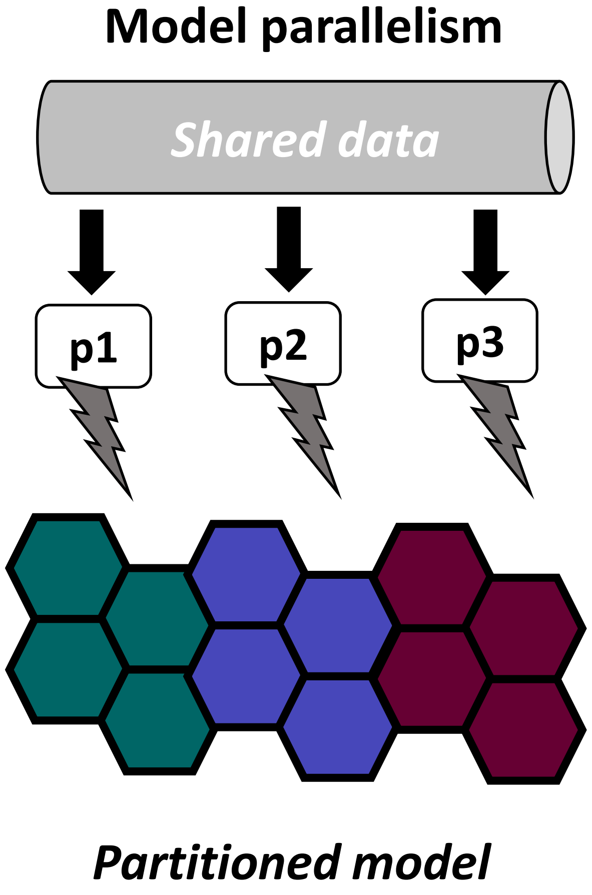
Applications in ML | Cross-validation



p1	Iteration 1	Train	Train	Train	Train	Test
p2	Iteration 2	Train	Train	Train	Train	Test
p3	Iteration 3	Train	Train	Train	Train	Train
p4	Iteration 4	Train	Train	Train	Train	Train
p5	Iteration 5	Train	Train	Train	Train	Train

Applications in ML | Model and data parallelism





Summary

- Basics
- Multiprocessing:
  - Starting a process, monitoring
- Sharing memory
  - Value, Array, Manager, Lock
- Further
  - Pool, CPU, GPU
- Application in ML
  - Hyperarameter optimization, Cross-validation, Model and Data parallelism

Q&A | Coding

Donations:

We are a registered non-profit and run on donations from people like you!

NongHyeop/농협은행 | 301 0275 2831 81
(코드서울)

https://github.com/CodeSeoul/machine-learning

https://discord.gg/HFknCs8