institut Curie

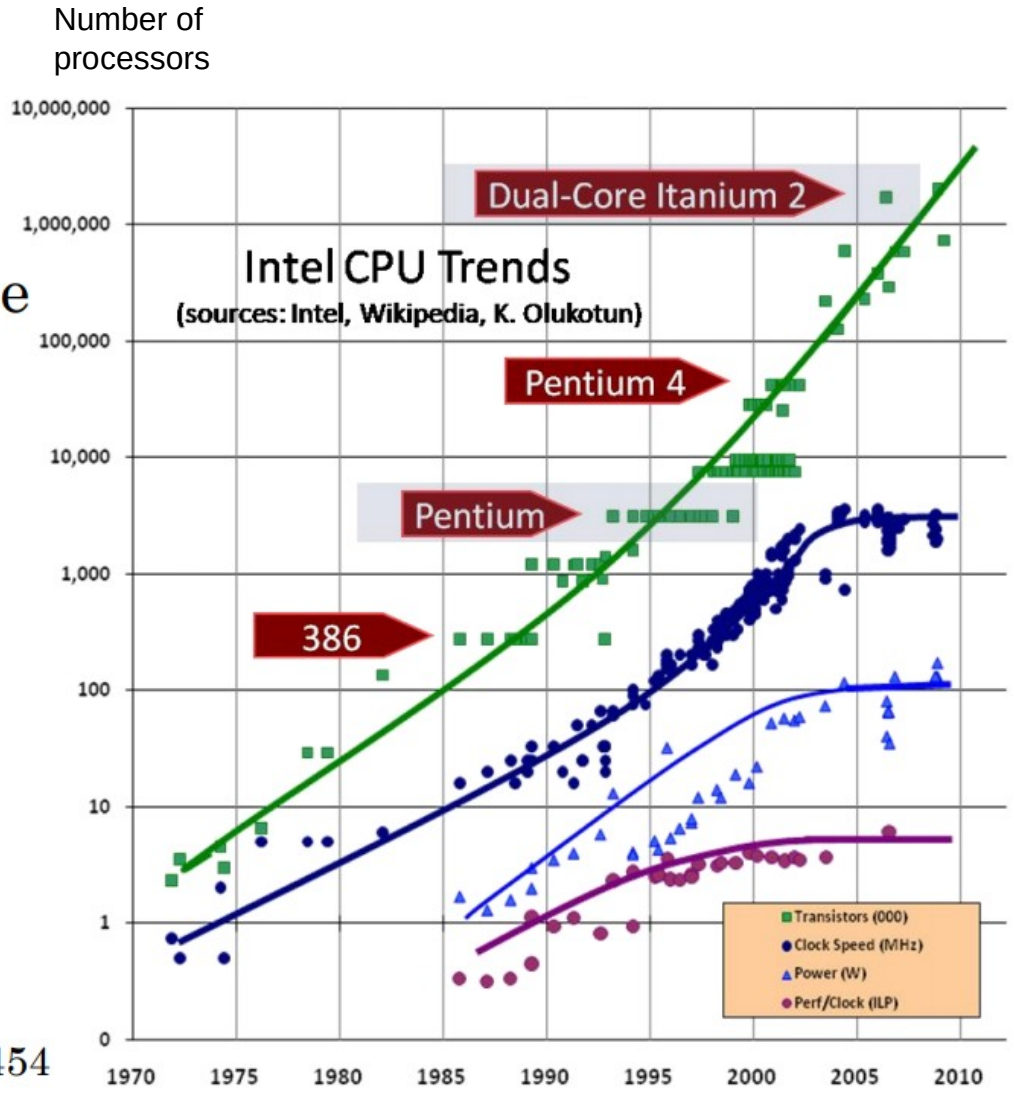
Inserm
La science pour la santé
From science to health

Multi Threading

Présentation Flash (5 min)

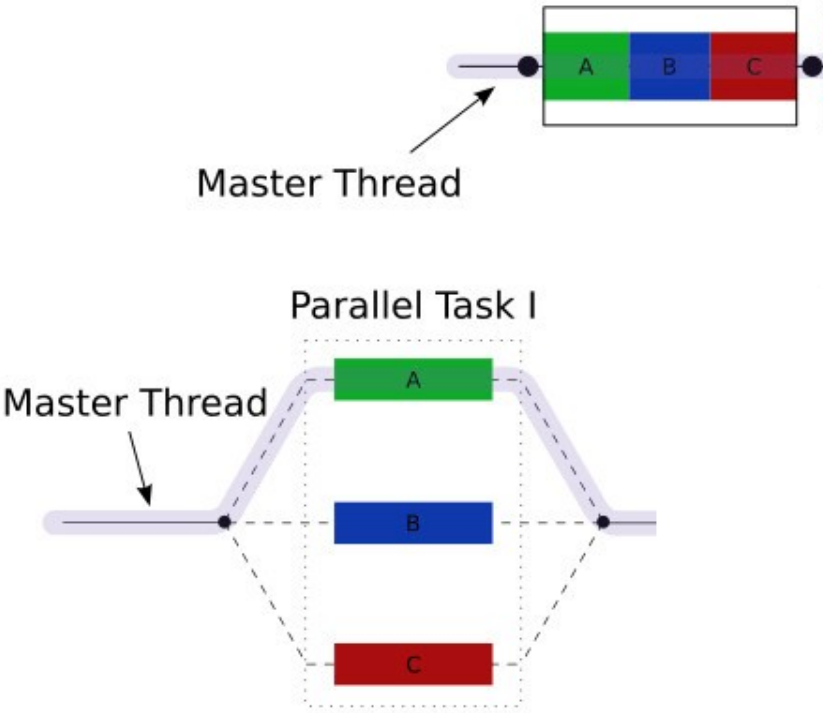
Multi threading (why ?)

- Modern Hardware is designed for thread level parallelism (TLP)

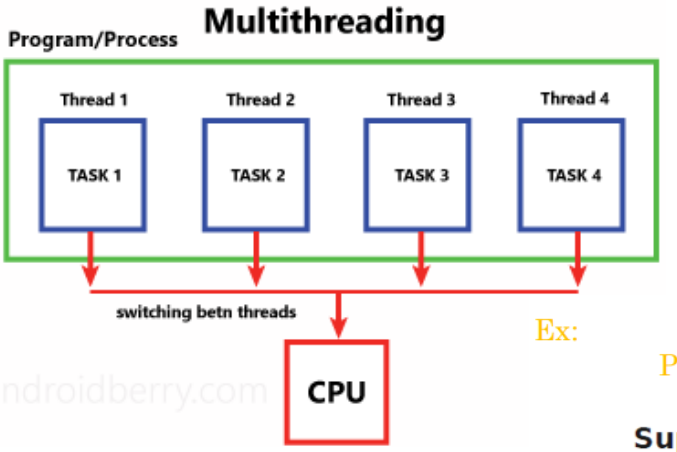


Source: Tom Ball - PPCP-54454

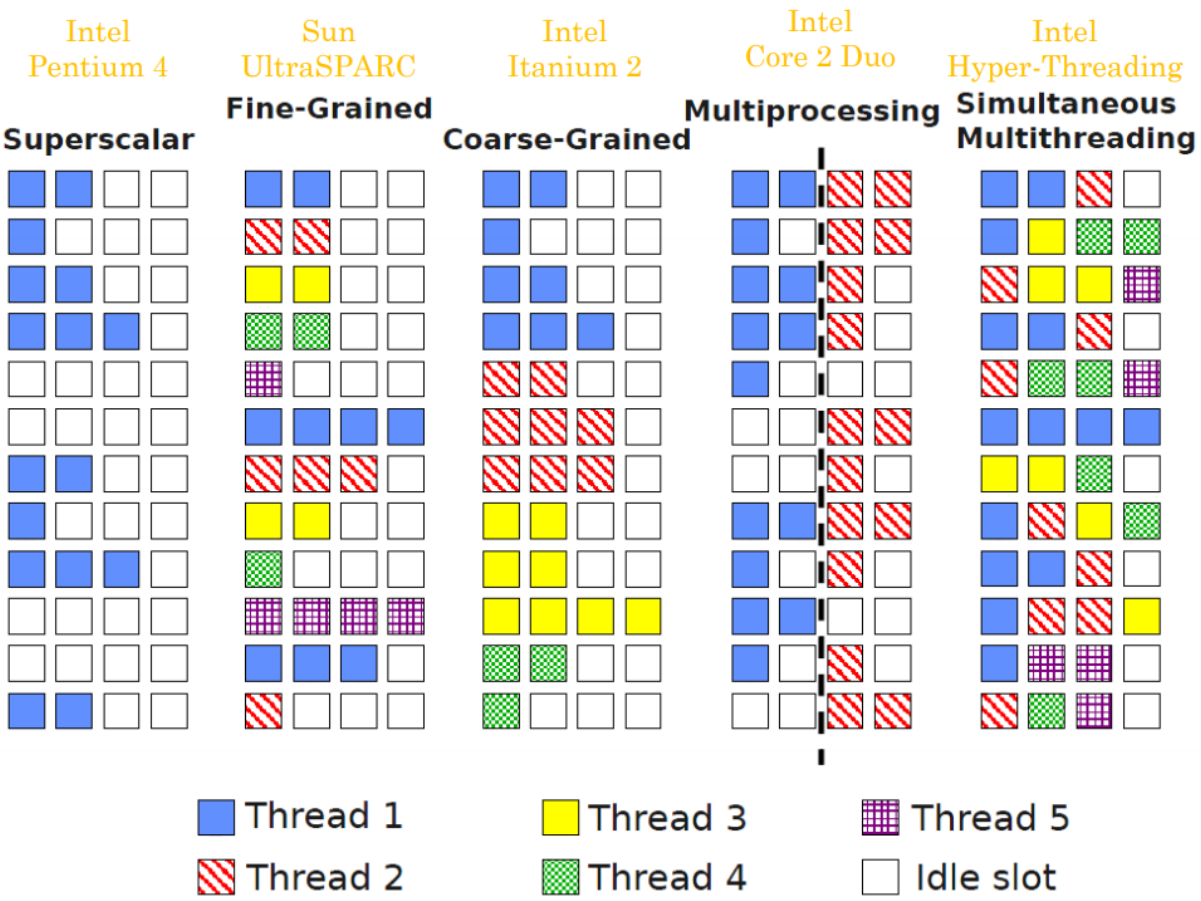
Multi threading



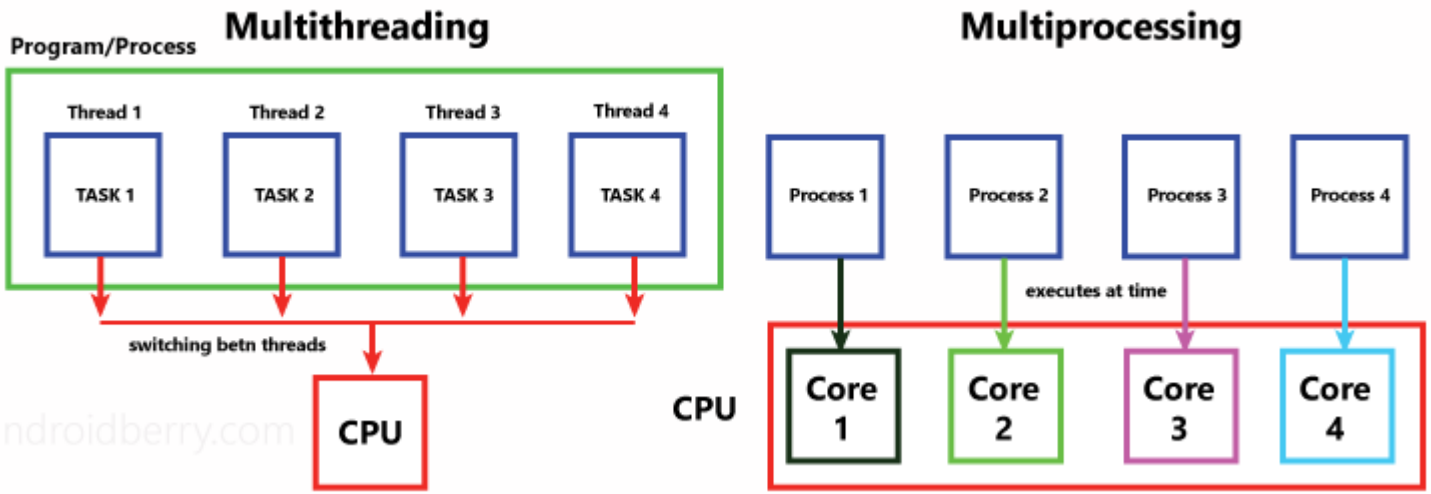
Multi threading



Ex:



Multi threading vs Multiprocessing



Python, Multi threading - processing

```
# setup output lists
output1 = list()
output2 = list()
output3 = list()
for j in range(0, 10):
    # calc individual parameter value
    parameter = j * offset
    # call the calculation
    out1, out2, out3 = calc_stuff(parameter = parameter)
    # put results into correct output list
    output1.append(out1)
    output2.append(out2)
    output3.append(out3)
```

```
pool = multiprocessing.Pool(4)
out1, out2, out3 = zip(*pool.map(calc_stuff, range(0, 10 * offset, offset)))
```

Note that this won't work in the interactive interpreter.

```
import concurrent.futures
import time, random                    # add some random sleep time
offset = 2                             # you don't supply these so
def calc_stuff(parameter=None):        # these are examples.
    sleep_time = random.choice([0, 1, 2, 3, 4, 5])
    time.sleep(sleep_time)
    return parameter / 2, sleep_time, parameter * parameter
def procedure(j):                      # just factoring out the
    parameter = j * offset             # procedure
    # call the calculation
    return calc_stuff(parameter=parameter)
def main():
    output1 = list()
    output2 = list()
    output3 = list()
    start = time.time()                # let's see how long this takes

    with concurrent.futures.ProcessPoolExecutor() as executor:
        for out1, out2, out3 in executor.map(procedure, range(0, 10)):
            # put results into correct output list
            output1.append(out1)
            output2.append(out2)
            output3.append(out3)
    finish = time.time()
    # these kinds of format strings are only available on Python 3.6:
    # time to upgrade!
    print(f'original inputs: {repr(output1)}')
    print(f'total time to execute {sum(output2)} = sum({repr(output2)})')
    print(f'time saved by parallelizing: {sum(output2) - (finish-start)}')
    print(f'returned in order given: {repr(output3)}')
if __name__ == '__main__':
    main()
```

```
$ python3 -m futuretest
original inputs: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
total time to execute 33 = sum([0, 3, 3, 4, 3, 5, 1, 5, 5, 4])
time saved by parallellizing: 27.68999981880188 sec
returned in order given: [0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

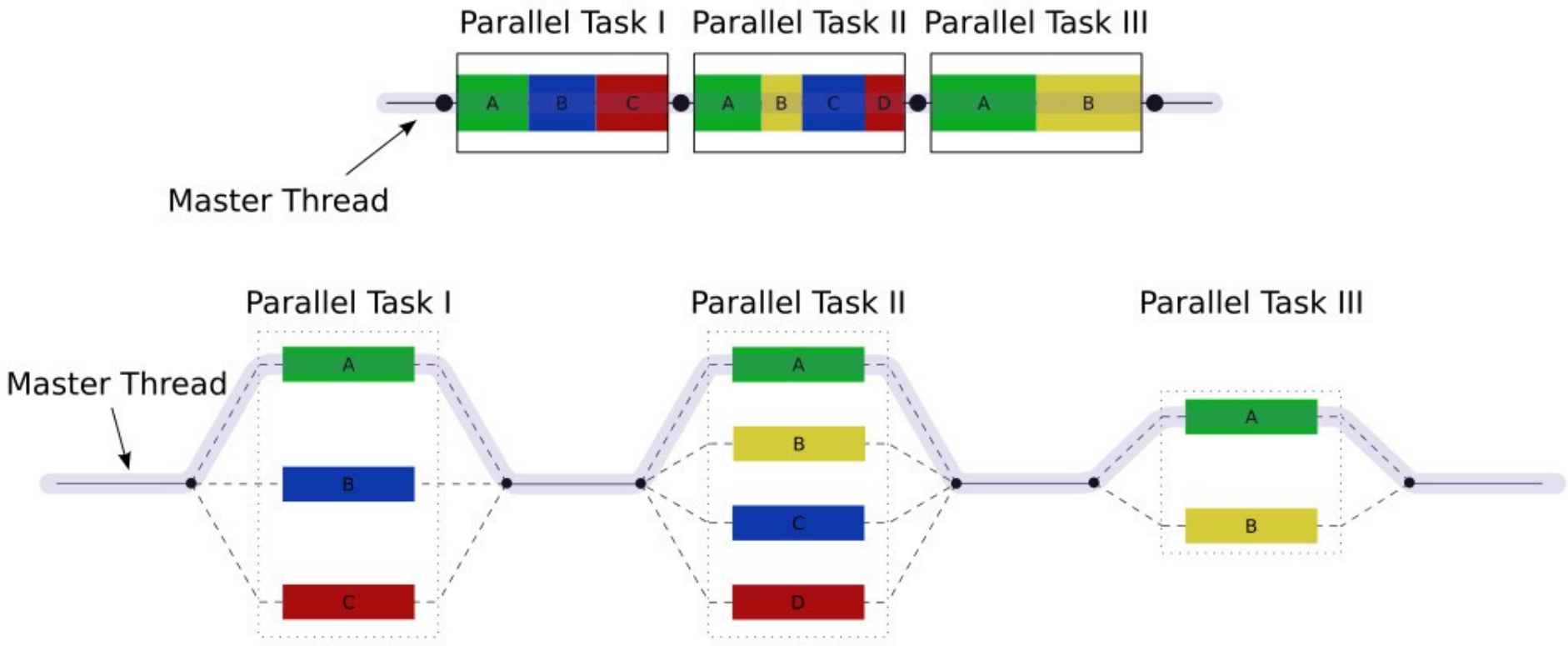
Multithreading
Now change ProcessPoolExecutor to ThreadPoolExecutor,
and run the module again:

```
$ python3 -m futuretest
original inputs: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
total time to execute 19 = sum([0, 2, 3, 5, 2, 0, 0, 3, 3, 1])
time saved by parallellizing: 13.992000102996826 sec
returned in order given: [0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
Now you have done both multithreading and multiprocessing!
```

multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local. It runs on both Unix and Windows.

multiprocessing est un paquet qui permet l'instanciation de processus via la même API que le module threading. Le paquet multiprocessing offre à la fois des possibilités de programmation concurrente locale. Il tourne à la fois sur les systèmes Unix et Windows.

Wainting for results



https://www.edureka.co/blog/what-is-mutithreading/