

Parallel Computing

So what's parallel computing?

Let's assume you have a cake



Option 1: eat each slice in order



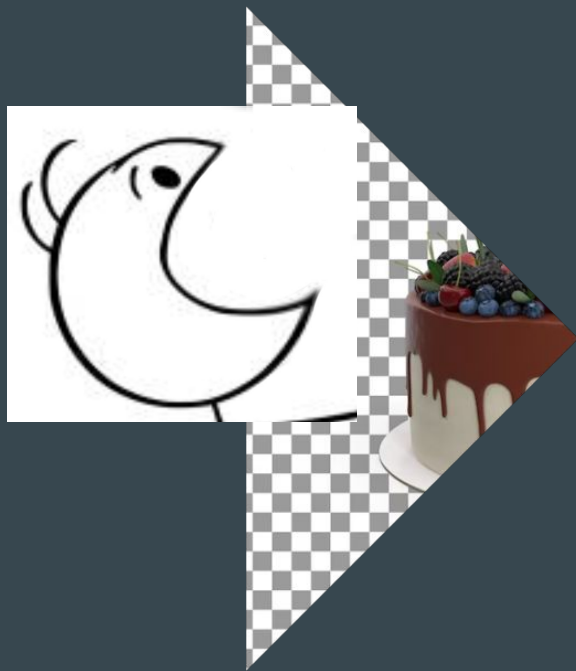
Option 1: eat each slice in order



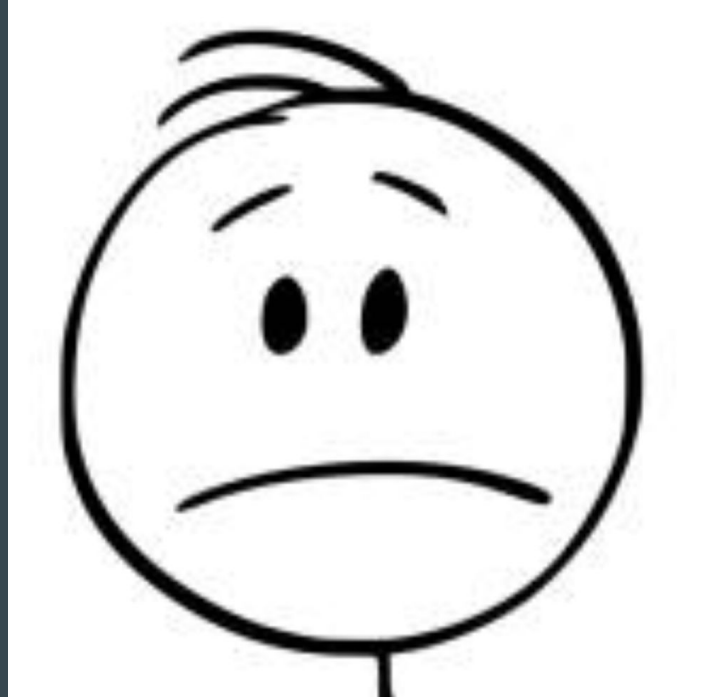
Option 1: eat each slice in order



Option 1: eat each slice in order



Boring, takes too much time



Boring, takes too much time

The old boring way - for loop

```
[26]: def EatSlice(cakeSlice):  
      return f"{cakeSlice}_eaten"
```

```
[9]: for cakeSlice_i, cakeSlice in enumerate(cake):  
      -----  
      print(f"Eating slice {cakeSlice_i}")  
      -----  
      cake[cakeSlice_i] = EatSlice(cakeSlice)  
      time.sleep(.5)
```

```
Eating slice 0  
Eating slice 1  
Eating slice 2  
Eating slice 3
```

```
[10]: cake
```

```
[10]: ['slice_0_eaten', 'slice_1_eaten', 'slice_2_eaten', 'slice_3_eaten']
```



Boring, takes too much time

The old boring way - map and lambda functions

```
] : EatSliceLambda = lambda cakeSlice: f"{cakeSlice}_eaten"
] : EatSliceLambda('Slice')
] : 'Slice_eaten'

] : chocolateCake = makeCake()
chocolateCake

] : ['slice_0', 'slice_1', 'slice_2', 'slice_3']

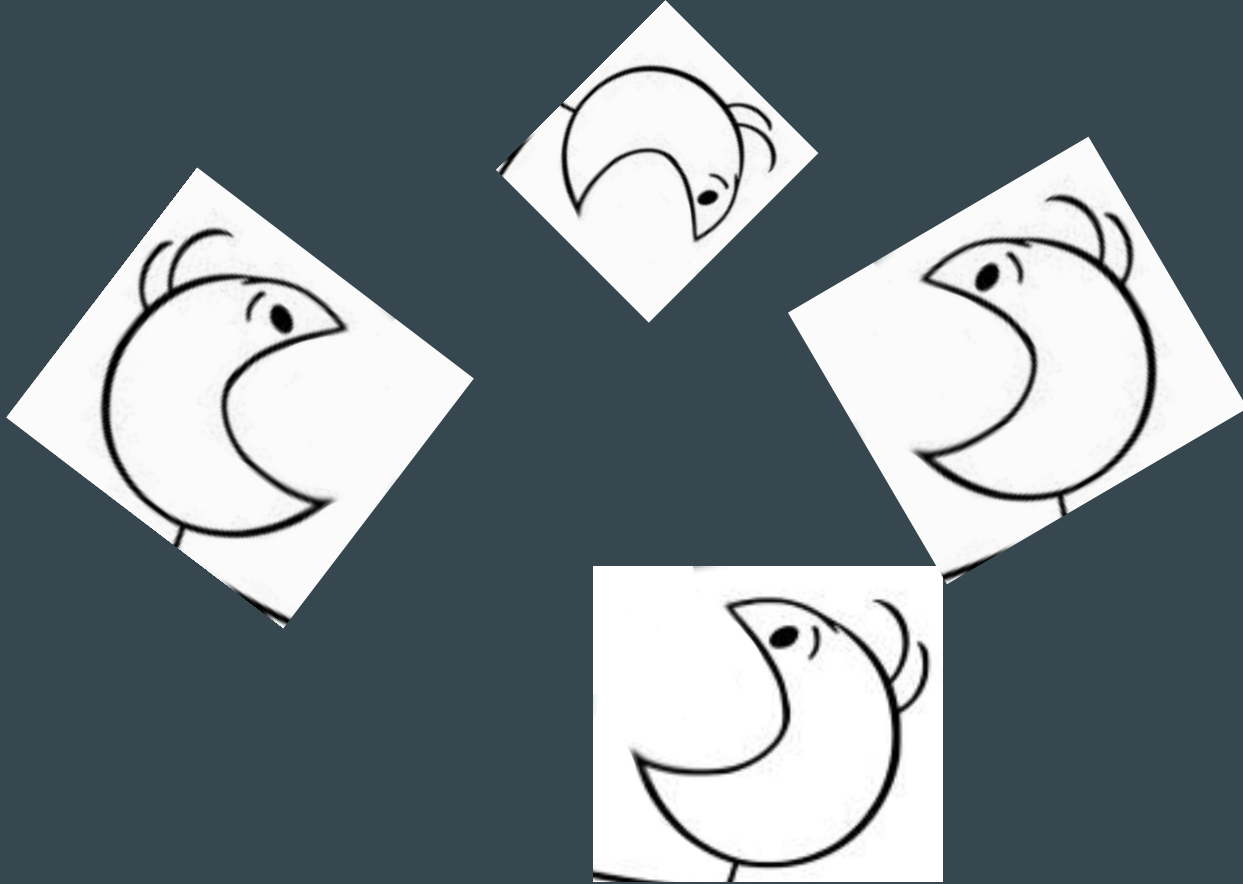
] : chocolateCake = list(map(EatSliceLambda, chocolateCake))
chocolateCake

] : ['slice_0_eaten', 'slice_1_eaten', 'slice_2_eaten', 'slice_3_eaten']
```

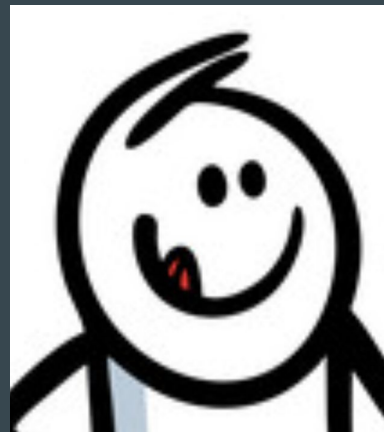
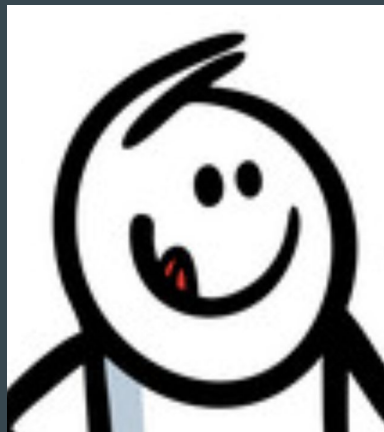
Option 2: Invite friends and eat them all at the same time



Option 2: Invite friends and eat them all at the same time



Fun, wayyyy faster



What happened?



They made a copy of the cake??!!??



And ate it???



But the original is untouched



Is the cake a lie????

But the original is untouched



```
[96]: import multiprocessing ## multiprocessing for within notebook, otherwise you can use multiprocessing
```

```
[24]: strawberryCake = makeCake()  
      strawberryCake
```


```
[24]: ['slice_0', 'slice_1', 'slice_2', 'slice_3']
```

```
[33]: def EatSliceParallel(cakeSlice_i):  
      strawberryCake[cakeSlice_i] = EatSlice(strawberryCake[cakeSlice_i])
```

```
[34]: with multiprocessing.Pool(2) as pool:  
      pool.map(EatSliceParallel, range(len(strawberryCake)))
```

```
[35]: strawberryCake
```

```
[35]: ['slice_0', 'slice_1', 'slice_2', 'slice_3']
```



But the original is untouched

If you need to modify the cake in the parallel process:

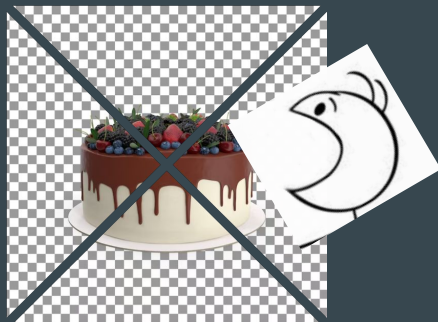
Shared memory

What happened? - 2

The return of what happened



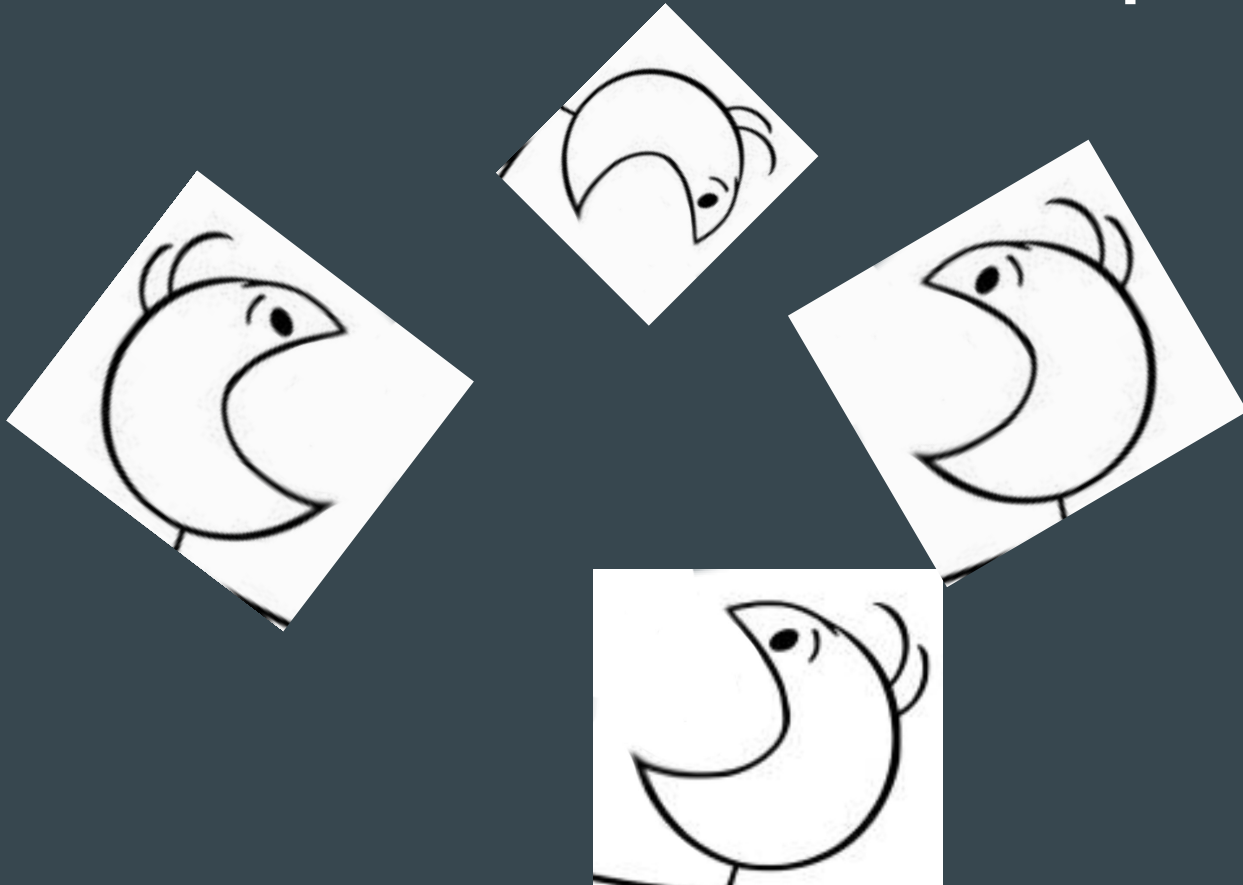
They made a copy of the cake



Ate the cake



And reassembled the modified part

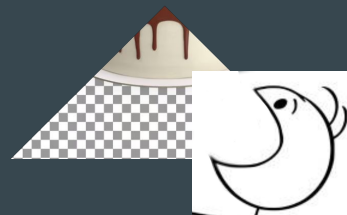
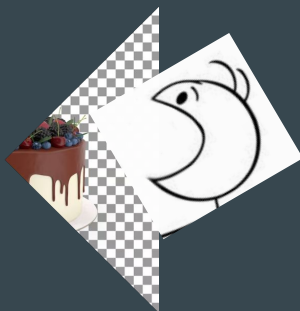
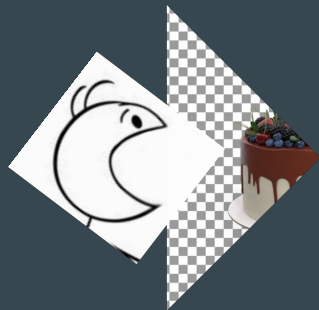


What happened? - 3

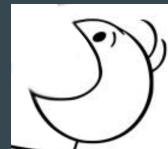
Revenge of what happened



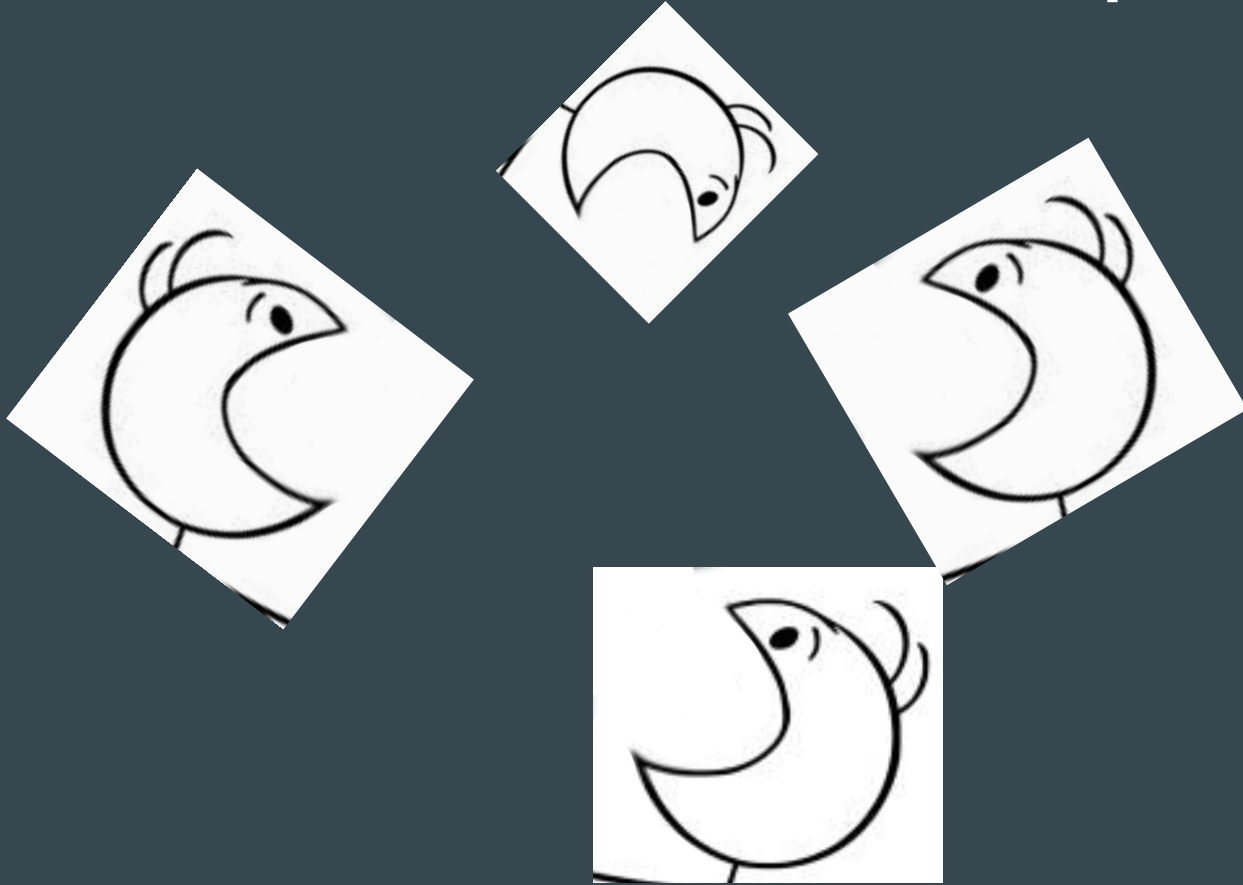
They made a copy of each slice



Ate the slice



And reassembled the modified parts



In general: think of how data is passed to the parallel process.

Memory can be expensive for large datasets.

**There is an overhead in starting a parallel process.
Think parallel if one task takes a long time.**

**A lot of python functions are already parallel.
E.G.: Numpy array computations are parallel**