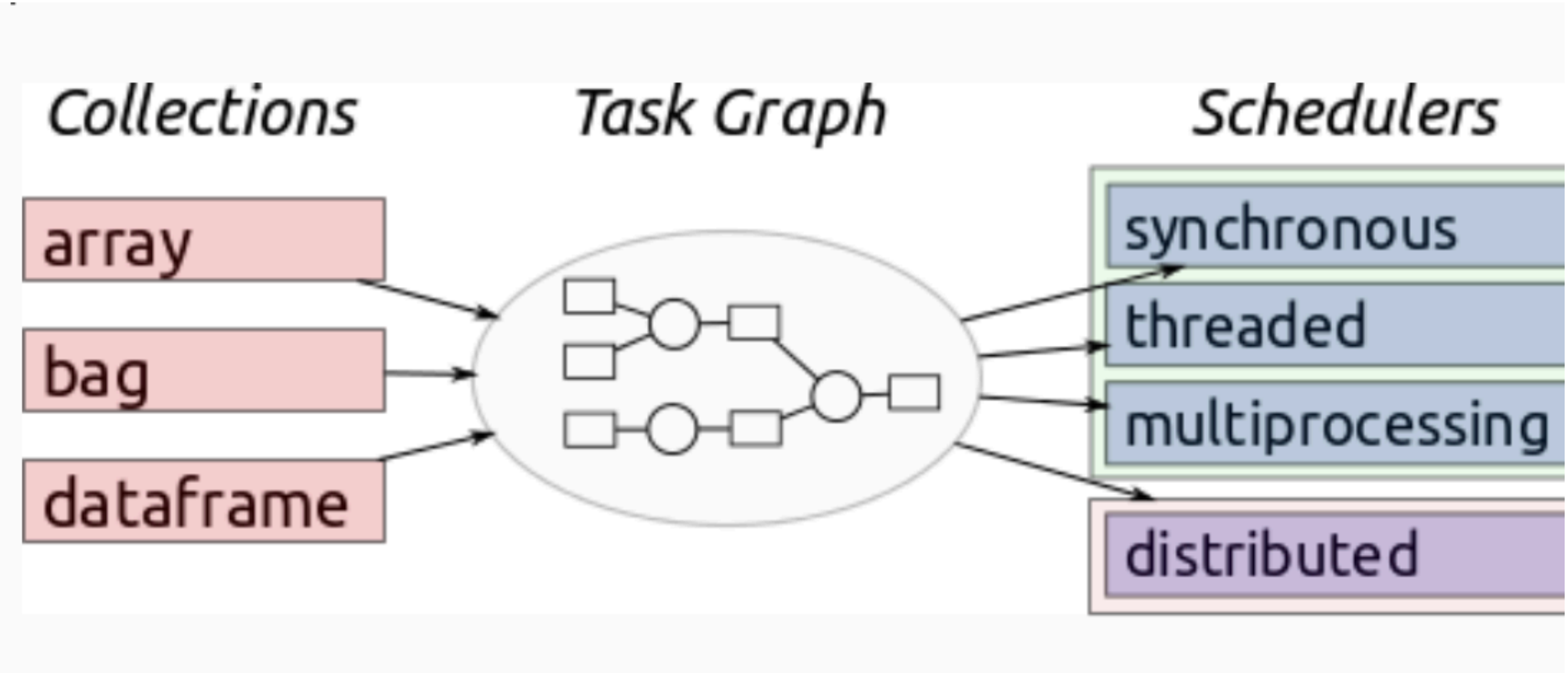
Dask integration

About Dask

- Transparently handles the brokerage of large datasets
- Transparently handles the distribution to workers
- Two relevant approaches:
    - “**processes**” (using multiprocessing.Pool —> iqoqomp)
        - Over ~80% of the use cases
    - “**distributed**” (needs to define cluster object and pass it to Dask)
        - Less than ~20% of the use cases
        - e.g. dask-yarn handles with clusters but we don’t need this extra layer

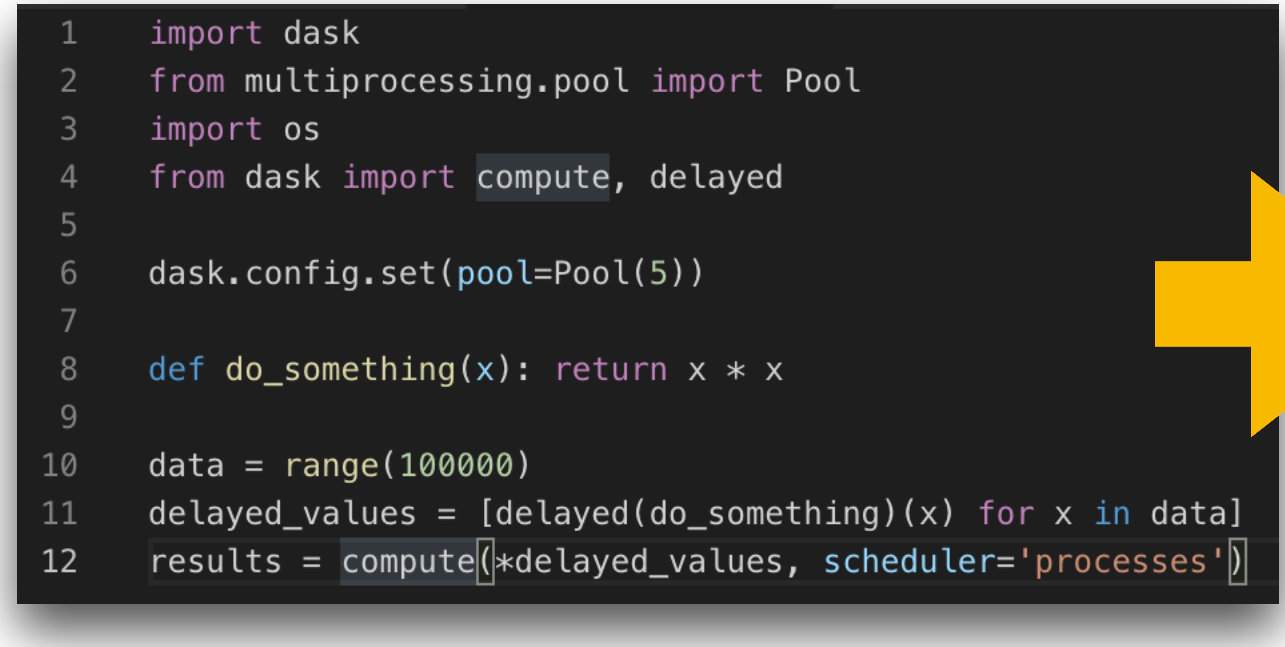
About Dask


Collections
Task Graph
Schedulers
array
bag
dataframe
synchronous
threaded
multiprocessing
distributed

Attempt with “processes”

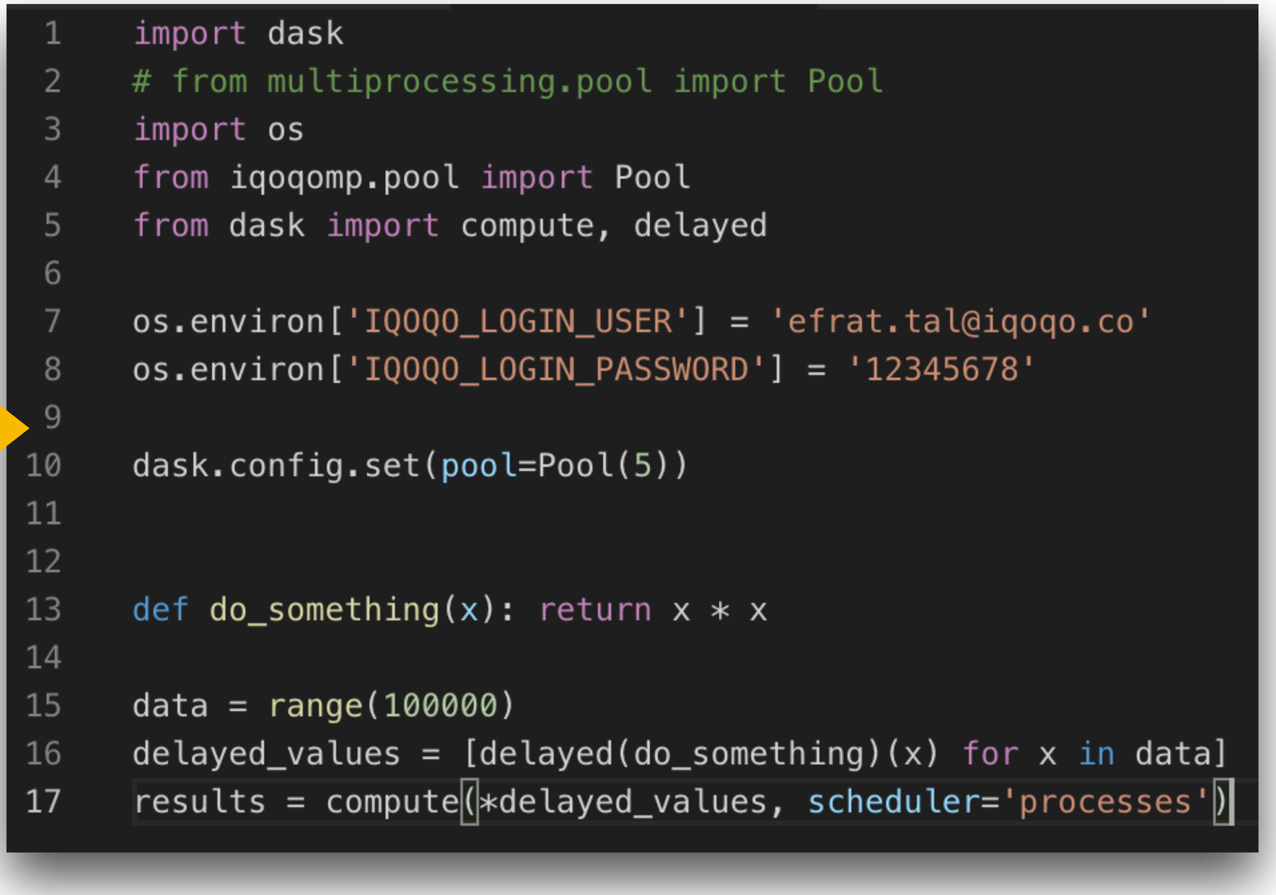
- Naively we could use iqoqomp instead of multiprocessing
- Problem is that desk has it builtin internally

Native multiprocessing+dask
(runs out of the box…)



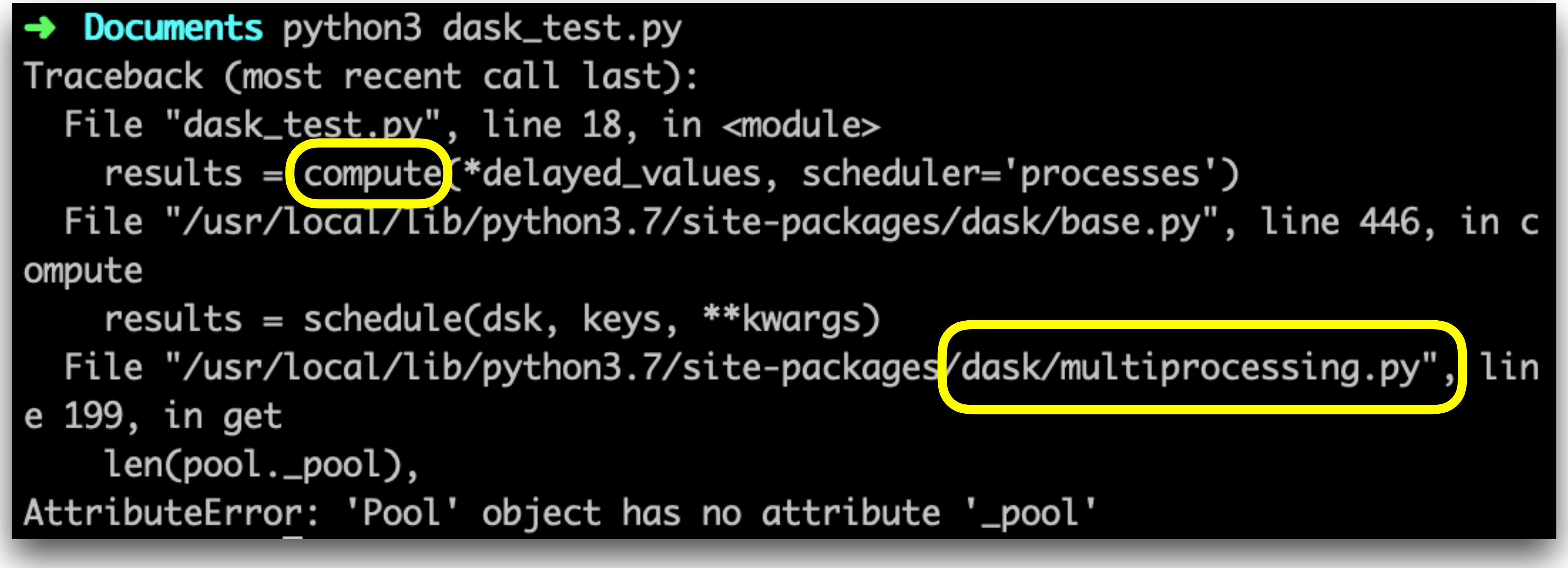
```python
import dask
from multiprocessing.pool import Pool
import os
from dask import compute, delayed

dask.config.set(pool=Pool(5))

def do_something(x): return x * x

data = range(100000)
delayed_values = [delayed(do_something)(x) for x in data]
results = compute(*delayed_values, scheduler='processes')
```

iqoqomp+dask
(fails - see next slide)



```python
import dask
# from multiprocessing.pool import Pool
import os
from iqoqomp.pool import Pool
from dask import compute, delayed

os.environ['IQOQO_LOGIN_USER'] = 'efrat.tal@iqoqo.co'
os.environ['IQOQO_LOGIN_PASSWORD'] = '12345678'

dask.config.set(pool=Pool(5))


def do_something(x): return x * x

data = range(100000)
delayed_values = [delayed(do_something)(x) for x in data]
results = compute(*delayed_values, scheduler='processes')
```

Dask “processes”



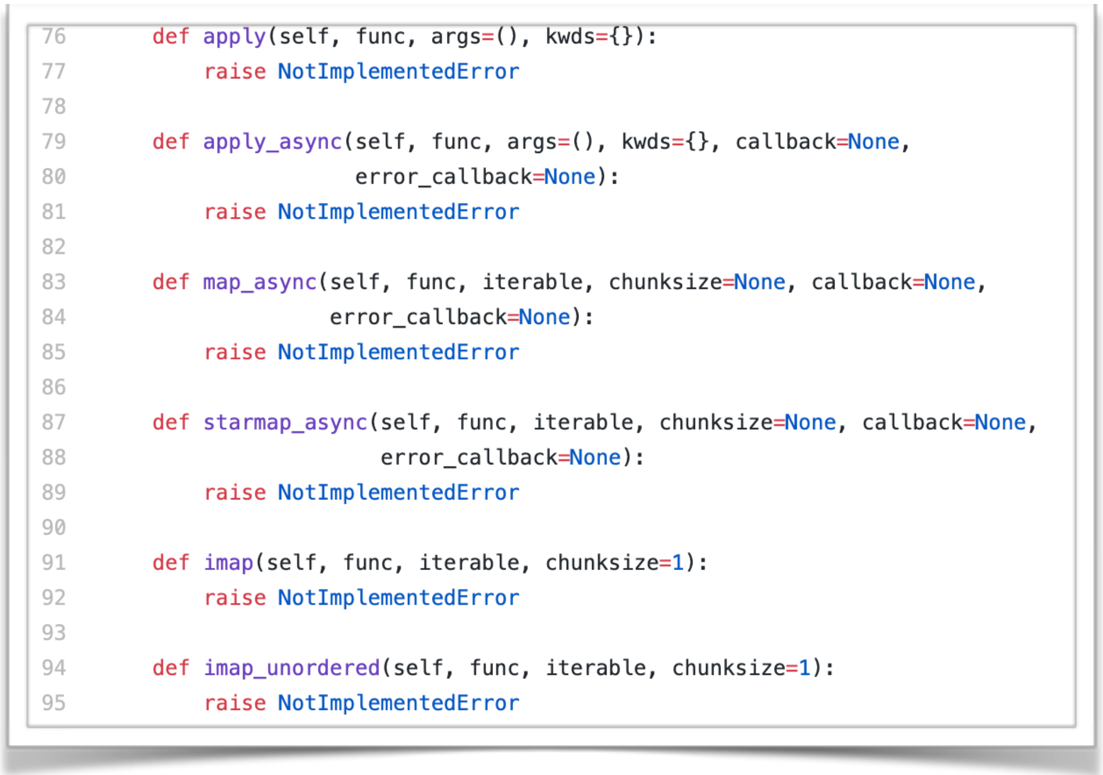
```
➜  Documents python3 dask_test.py
Traceback (most recent call last):
  File "dask_test.py", line 18, in <module>
    results = compute(*delayed_values, scheduler='processes')
  File "/usr/local/lib/python3.7/site-packages/dask/base.py", line 446, in c
ompute
    results = schedule(dsk, keys, **kwargs)
  File "/usr/local/lib/python3.7/site-packages/dask/multiprocessing.py", lin
e 199, in get
    len(pool._pool),
AttributeError: 'Pool' object has no attribute '_pool'
```

- This image just shows the beginning of the process, i.e. even if I add the _pool object to the class, things fail downstream

- Will need to hack Dask or *fully* implement iqoqomp Pool class as the Pool class of multiprocessing.py (at least)

- Not shown here but also the multithreaded option is impossible

iqoqomp.Pool



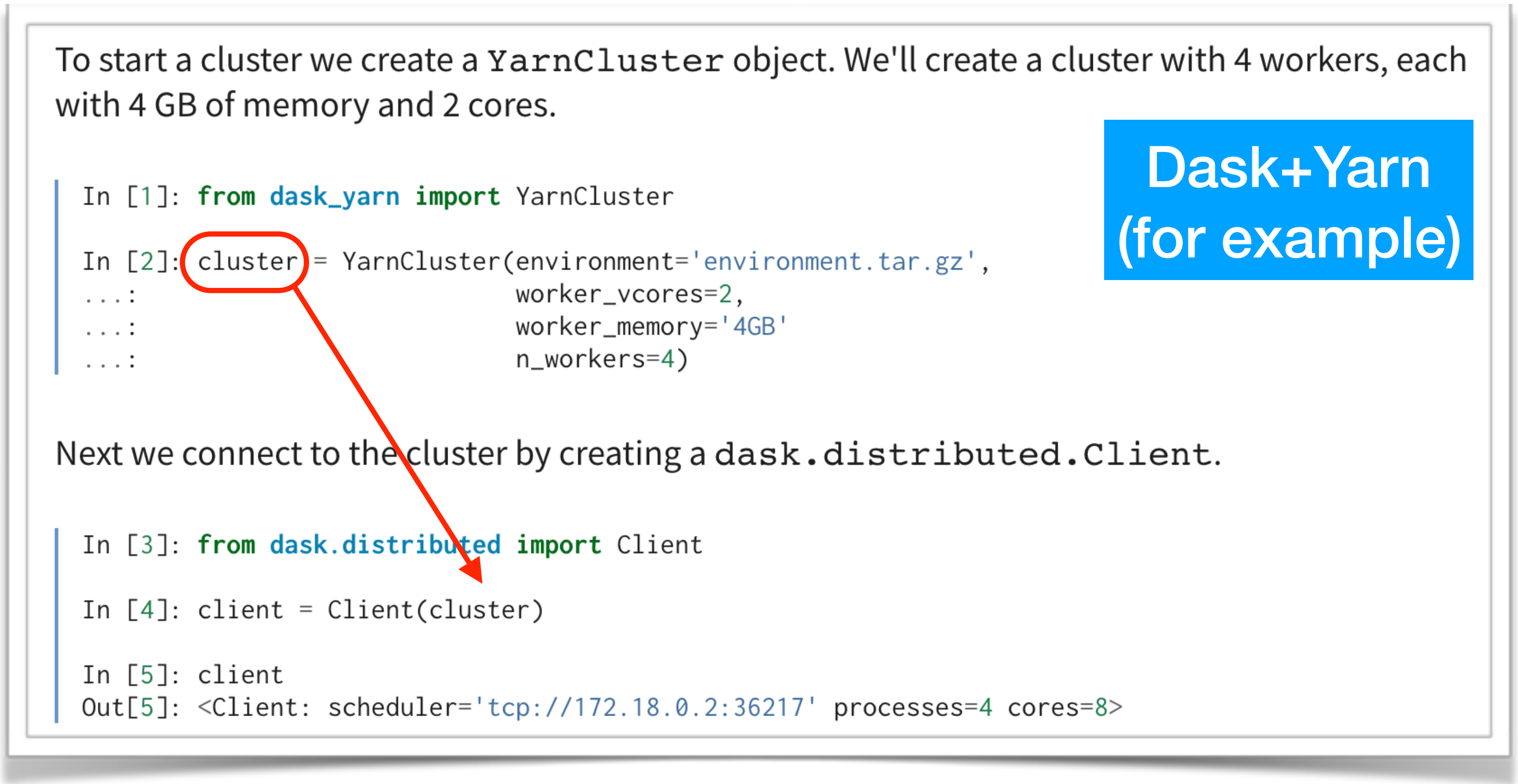
```
    def apply(self, func, args=(), kwds={}):
        raise NotImplementedError

    def apply_async(self, func, args=(), kwds={}, callback=None,
                    error_callback=None):
        raise NotImplementedError

    def map_async(self, func, iterable, chunksize=None, callback=None,
                  error_callback=None):
        raise NotImplementedError

    def starmap_async(self, func, iterable, chunksize=None, callback=None,
                      error_callback=None):
        raise NotImplementedError

    def imap(self, func, iterable, chunksize=1):
        raise NotImplementedError

    def imap_unordered(self, func, iterable, chunksize=1):
        raise NotImplementedError
```

- Many features are not implemented (on purpose)
- Using “processes” approach, Dask.compute() is expecting a complete Pool object <u>and</u> it relies on the internal copy of multiprocessing which has much more functionalities
  - This approach is really single-machine-oriented and many things are hardcoded there for that reason
  - It could be solved with some adaptations on their side, **<u>but</u>** this is why they have the “distributed” architecture, so I doubt they will accept any change we may come with

“distributed”

- It is not the most frequent use case because it is “more complex” for random users
- Need direct access to the cluster
  - Dask provides some interfaces with Amazon / Google /… clouds
  - Example with Yarn below (irrelevant for us but just to make a point)
    - It provides the entire machinery that we provide, including the definition of the cluster
- If we go that way, we need to write a wrapper that returns the iqoqo “cluster” as a one-liner

Estimate the implementation to require ~4 weeks at least



Dask+Yarn (for example)

To start a cluster we create a `YarnCluster` object. We'll create a cluster with 4 workers, each with 4 GB of memory and 2 cores.

```
In [1]: from dask_yarn import YarnCluster

In [2]: cluster = YarnCluster(environment='environment.tar.gz',
   ...:                       worker_vcores=2,
   ...:                       worker_memory='4GB'
   ...:                       n_workers=4)
```

Next we connect to the cluster by creating a `dask.distributed.Client`.

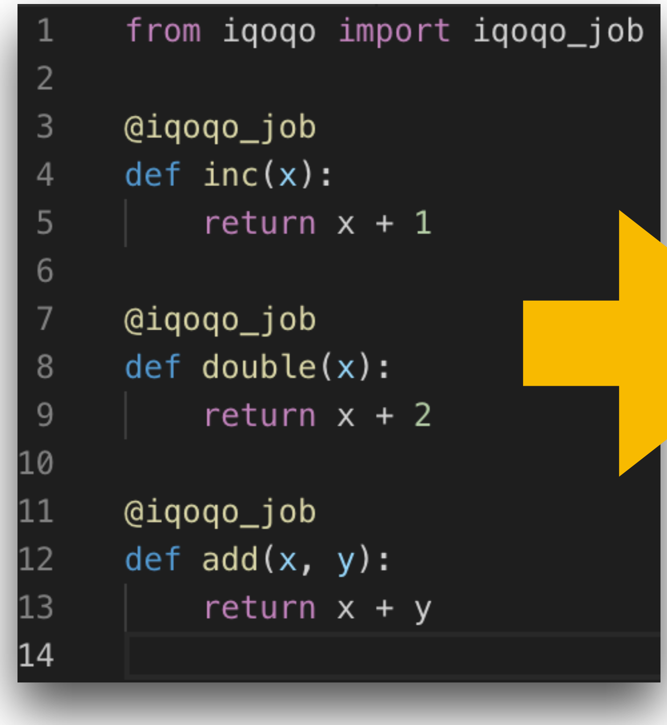
```
In [3]: from dask.distributed import Client

In [4]: client = Client(cluster)

In [5]: client
Out[5]: <Client: scheduler='tcp://172.18.0.2:36217' processes=4 cores=8>
```

“delayed”

- One more very esoteric use case
    - strips-off most of Dask advantages
    - when the usual data structures cannot be used
    - user has some control on the parallelisation
    - useful for strange data structures and for complex inter-process dependencies
- Small demo using the iqoqo sdk demonstrates the point but
    - we have to force no dependencies
    - We have to force non-Dask data structures
- This approach has no advantage on simply using iqoqomp and it does not exploit any of the main features of Dask.

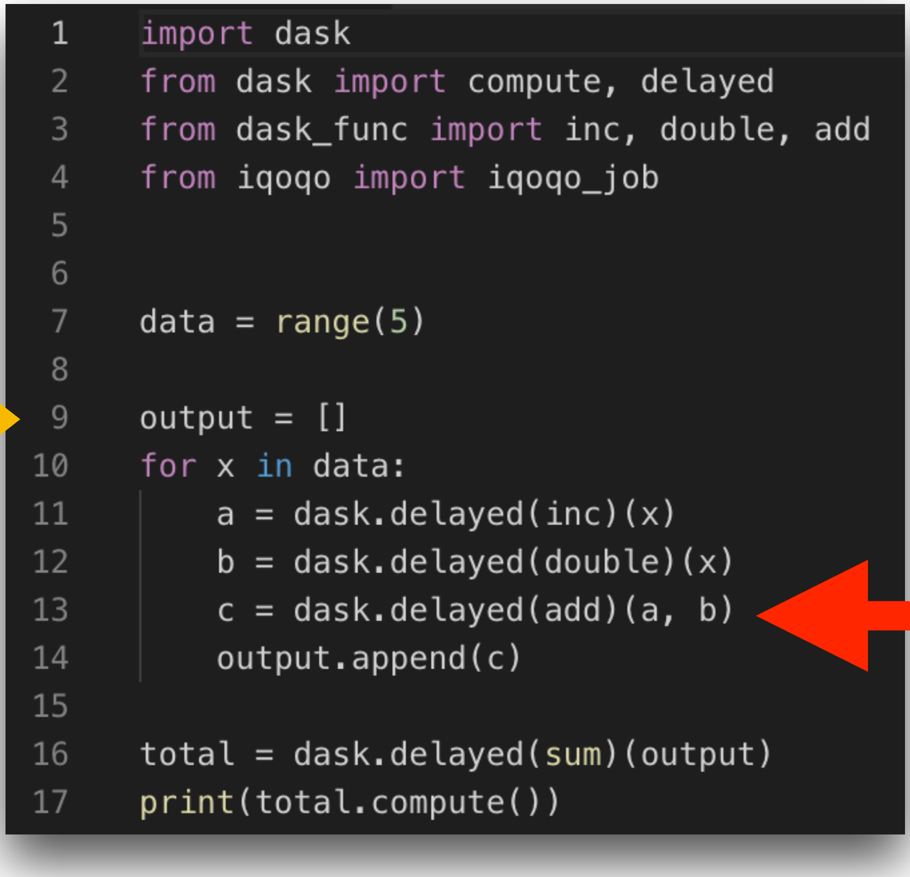
“delayed”

dask_func.py (just functions)

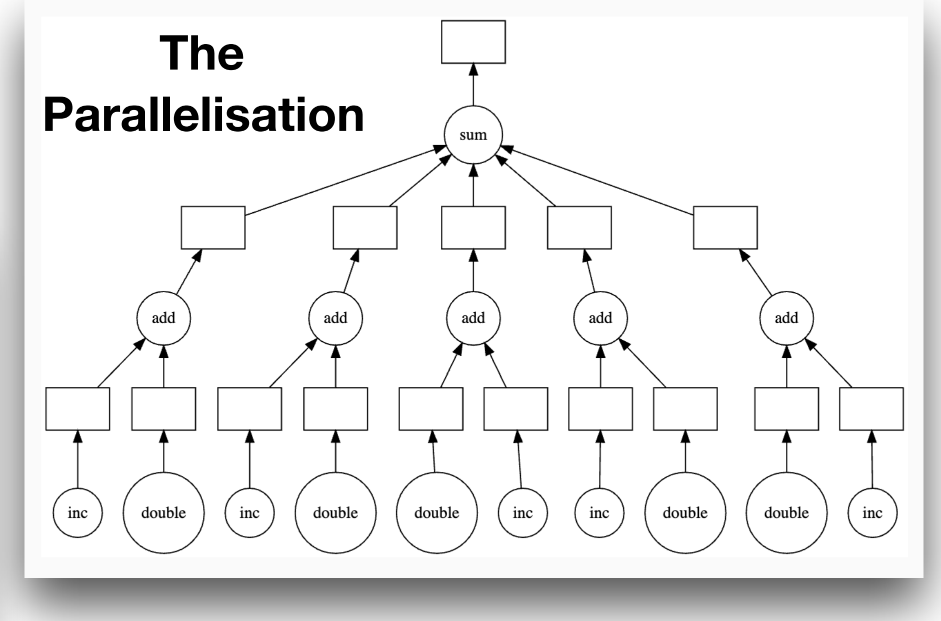


```python
from iqoqo import iqoqo_job

@iqoqo_job
def inc(x):
    return x + 1

@iqoqo_job
def double(x):
    return x + 2

@iqoqo_job
def add(x, y):
    return x + y

```

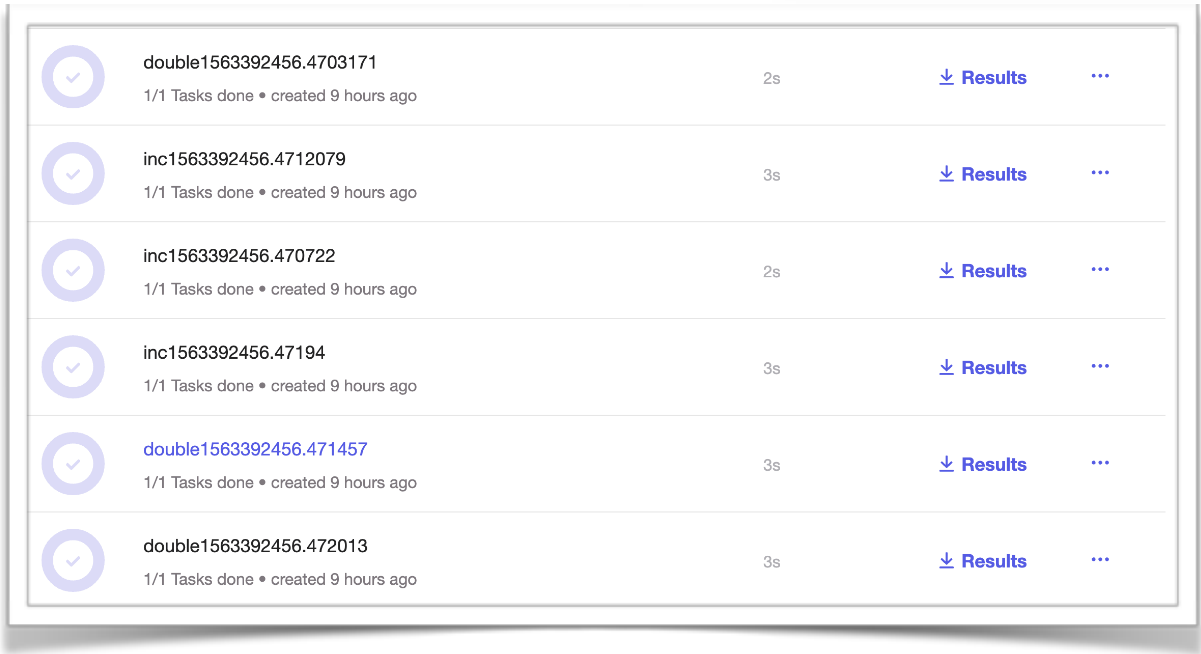
dask_delayed.py (steering)



```python
import dask
from dask import compute, delayed
from dask_func import inc, double, add
from iqoqo import iqoqo_job


data = range(5)

output = []
for x in data:
    a = dask.delayed(inc)(x)
    b = dask.delayed(double)(x)
    c = dask.delayed(add)(a, b)
    output.append(c)

total = dask.delayed(sum)(output)
print(total.compute())
```



The dependencies wont work in this example on ANY (remote) cluster



double1563392456.4703171
1/1 Tasks done • created 9 hours ago 2s Results

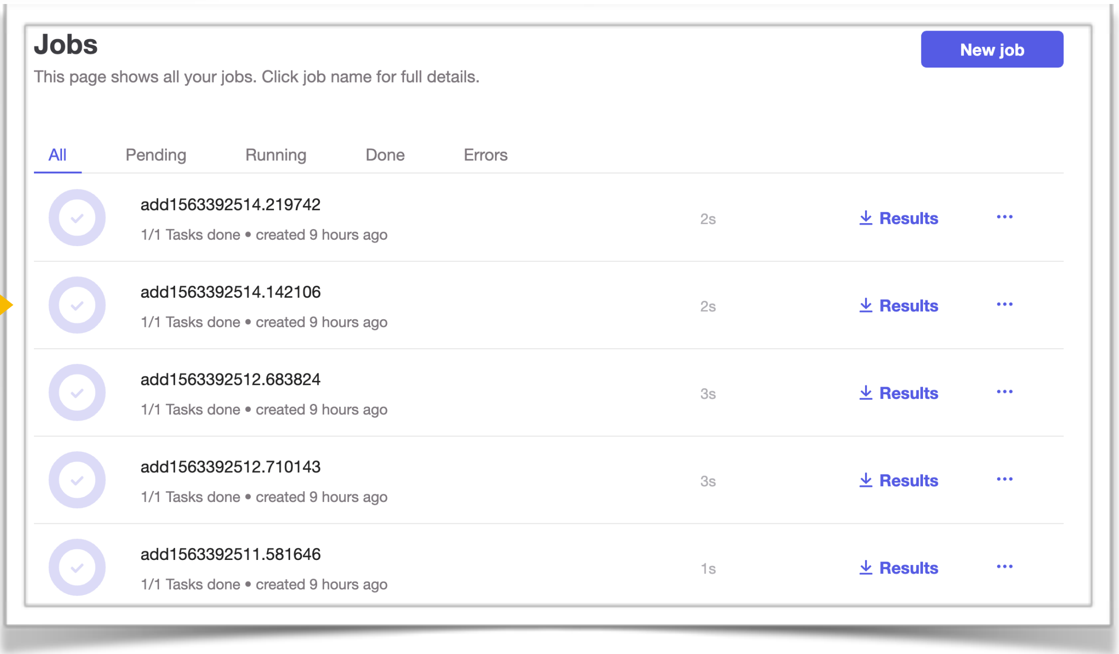
inc1563392456.4712079
1/1 Tasks done • created 9 hours ago 3s Results

inc1563392456.470722
1/1 Tasks done • created 9 hours ago 2s Results

inc1563392456.47194
1/1 Tasks done • created 9 hours ago 3s Results

double1563392456.471457
1/1 Tasks done • created 9 hours ago 3s Results

double1563392456.472013
1/1 Tasks done • created 9 hours ago 3s Results



Jobs

This page shows all your jobs. Click job name for full details.

add1563392514.219742
1/1 Tasks done • created 9 hours ago 2s Results

add1563392514.142106
1/1 Tasks done • created 9 hours ago 2s Results

add1563392512.683824
1/1 Tasks done • created 9 hours ago 3s Results

add1563392512.710143
1/1 Tasks done • created 9 hours ago 3s Results

add1563392511.581646
1/1 Tasks done • created 9 hours ago 1s Results