Università degli studi di Padova

Management and Analysis of Physics Datasets

DASK a simple way to use and implement Cluster and HPC

Stefano Campese

Why to use Cluster and HPC?

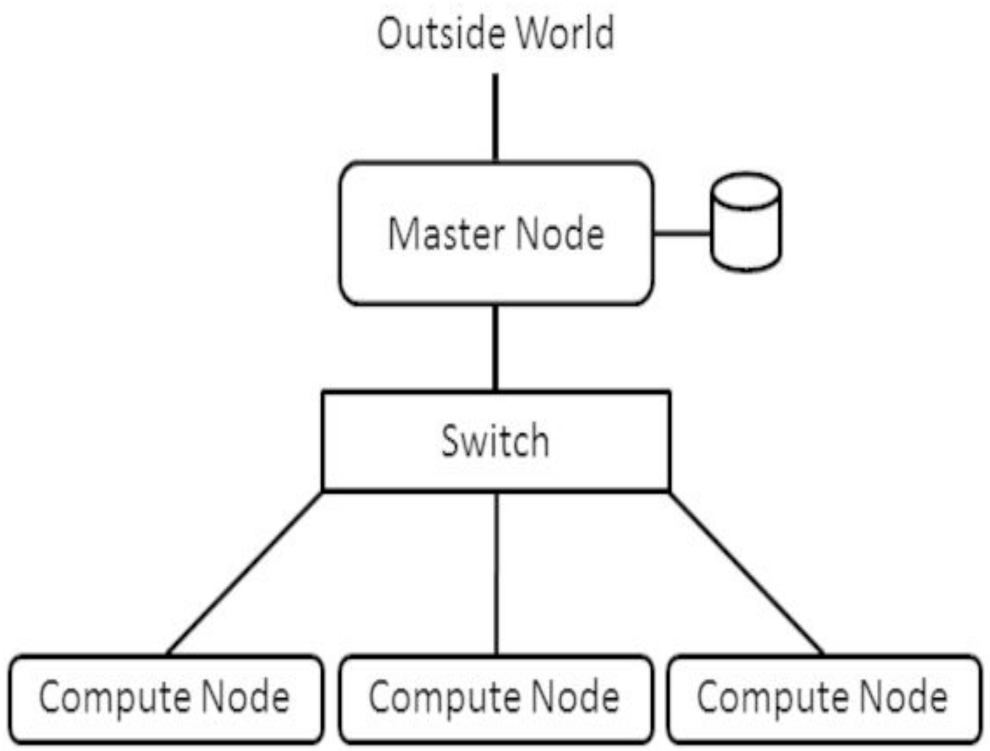
- **Huge data (*Big data*)**
- **Complex / long computations:**

  some computations are not affordable with a PC/Laptop
- **Intensive processing**
- **Specific designed Hardware**
- **Reducing the computation costs**
- **Reliability**:

  if one node goes down, the system still continues to work
- **Scalability**:

  memory, processor and nodes can be added as

  resources demand increases



Cons of Cluster and HPC?

- **Cluster networking**:

  networks must have high speed and must be reliable
- **Cluster configuration**:

  configurations can be complicated and required a lot of knowledge and time
- **Authentication and shared resource**:

  Some processes/data should be available only for some users, but the HPC/Cluster resources are shared amongst all users
- **Programming**:

  programming techniques are really different (asynchronous, graph programming)
- **Timing**:

  different nodes of the HPC/Cluster may have different computation power with different execution times

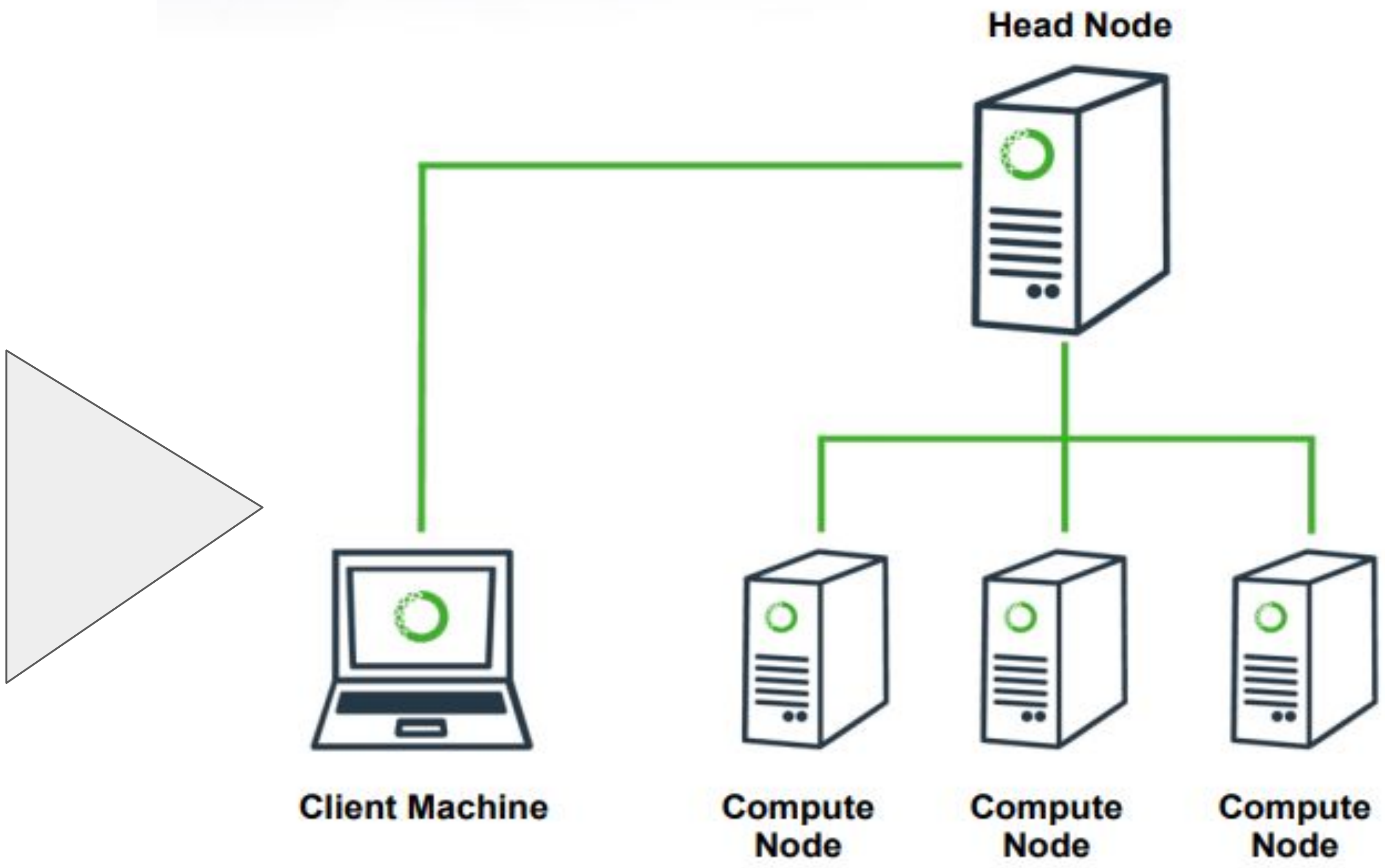
Example of architecture

Big data processing

Complex data modelling

Series of tasks

Data analysis

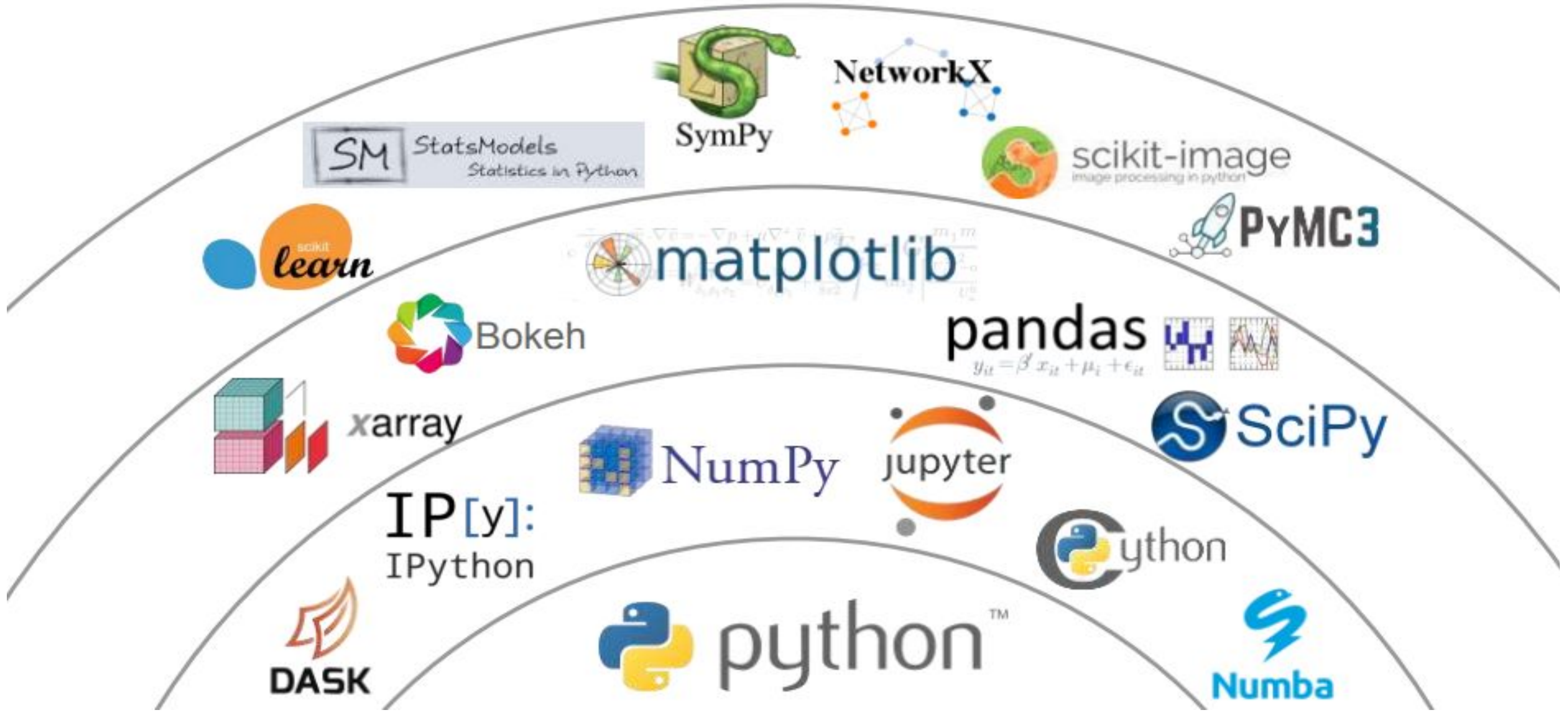


Don't panic!

Working with cluster and HPC may be complex, but some useful instruments help us!

Why Python?

All the main scientific programs and development tools are written in Python thanks to its flexibility and simplicity....



Why DASK?

- Intended and written to be flexible and simple in conjunction with all Python scientific packages (numpy, pandas, SKLearn, etc..)
- Small/none configuration knowledges needed
- Designed to parallelize and to distribute all Python ecosystem
- Simplifies to use HCP and Clusters
- Can scale from multicore Pc to HPC and Cluster with thousands of nodes
- Designed to work with Big data and distributed data
- Simplifies to work with constraints on shared resources

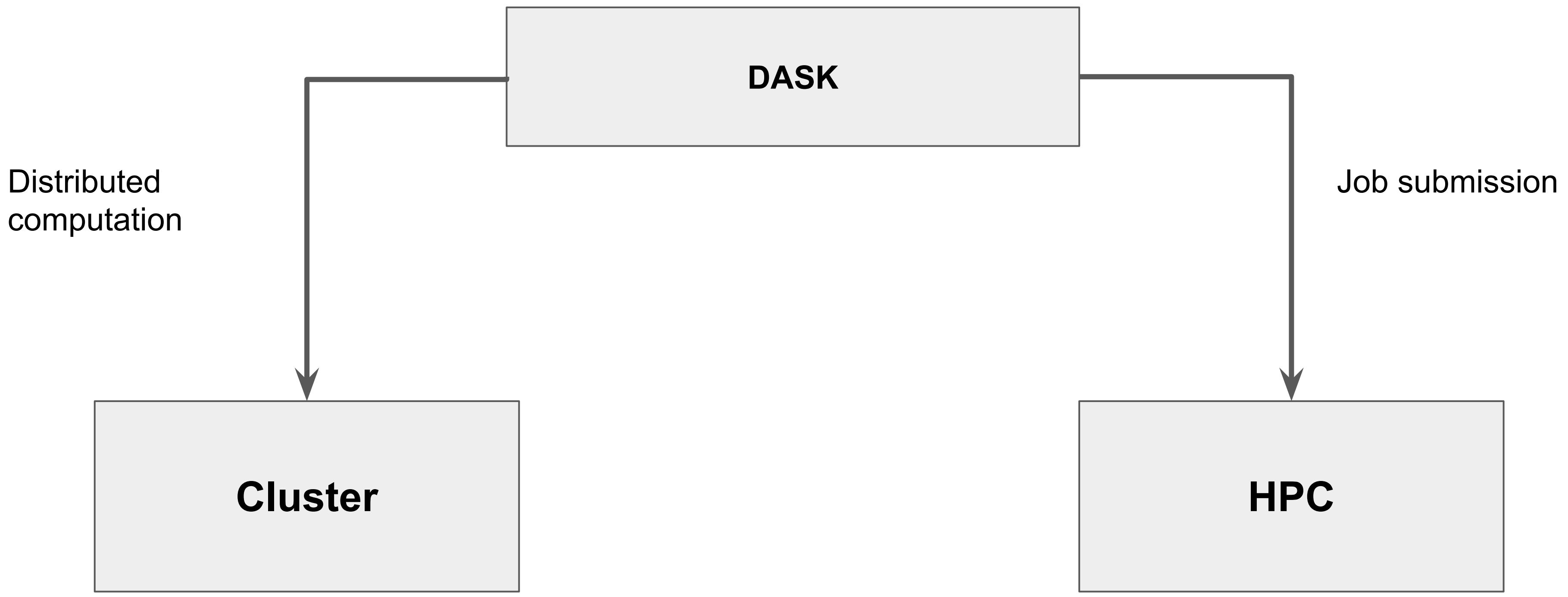
Why DASK?

- Perform large computation with less memory footprint
- Algorithms specifically designed to work with chunks of data
- Management of failures nodes
- Easy to add nodes to on the fly
- Real time dashboard where monitoring what is happening on the cluster

Limitations of DASK?

- Computation limitations due to the Python language
- Assignation of the jobs to the workers is always optimal (a lot of research here…)
- Assumes that functions and data are both serializable
- In case of failure Dask re-run your code multiple times
- Real time dashboard where monitoring what is happening on the cluster

How DASK can be used?



How DASK works?

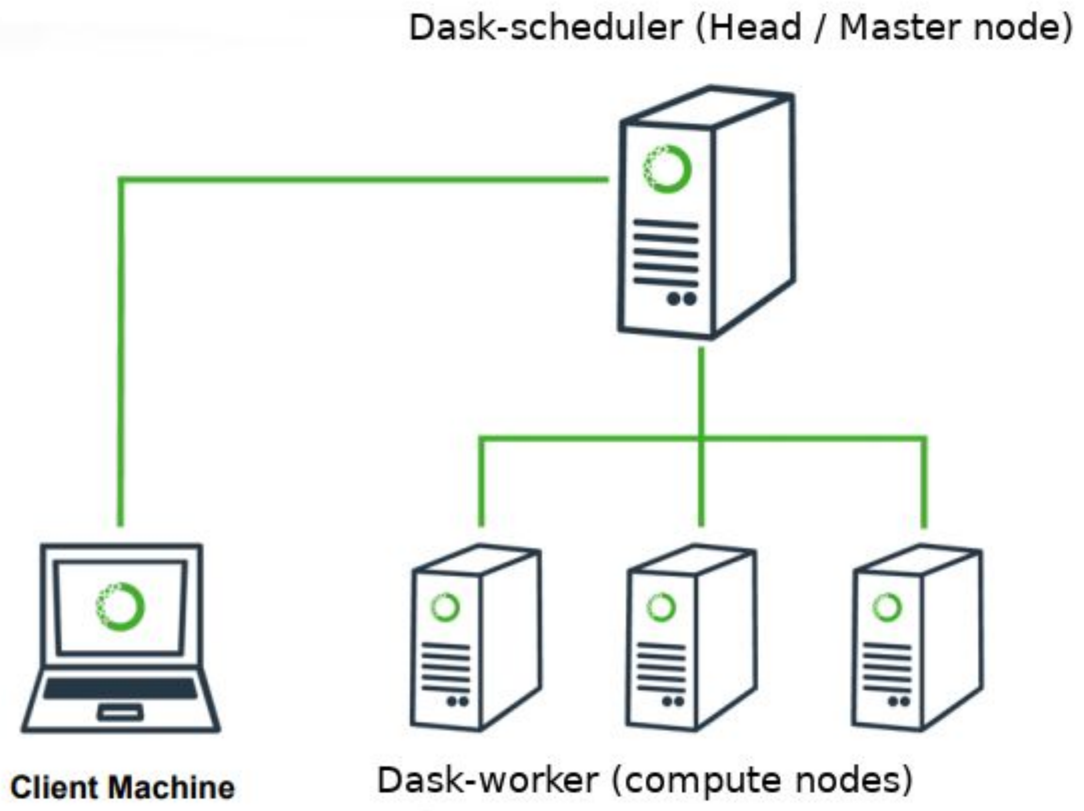
Create a cluster with DASK is simple!

- **Client Machine**: is the machine from which an user has submitted a job or has distributed a program.

- **Dask-scheduler**: is the head/master node of a cluster/HPC and decides how to schedule the execution of the processes on the **Pool of resources.**

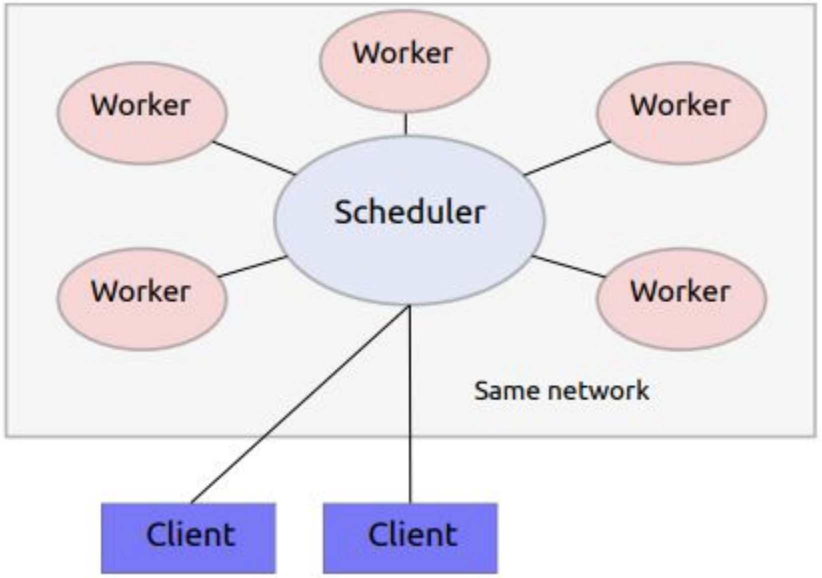
- **Dask-worker**: represents a “computing node” of a Cluster/HPC that receives the task that should be executed. A **Pool of resources** is a set of these nodes.

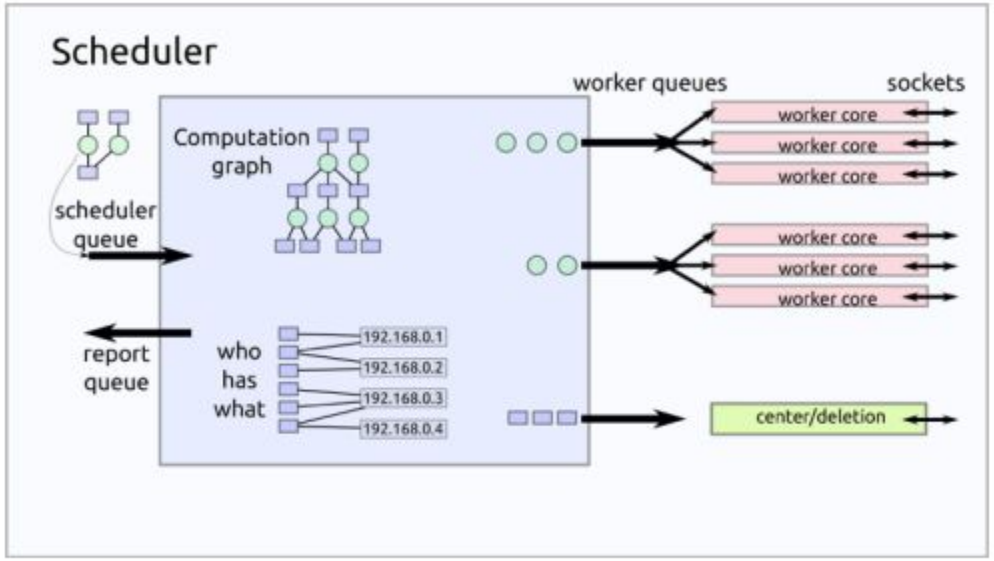
DASK *follows* the **Push Model:**
each job is pushed to the pool of the computational resources


Dask-scheduler (Head / Master node)
Client Machine
Dask-worker (compute nodes)

How it works?

- DASK functioning is based on the usage of DAG (*Directed acyclic graphs*).
- Each time a client machine submits a task, the Dask-scheduler creates/updates a graph (called *queue*).
- Once the building of the computation graph is terminated, each node of this DAG is processed by the dask-workers.





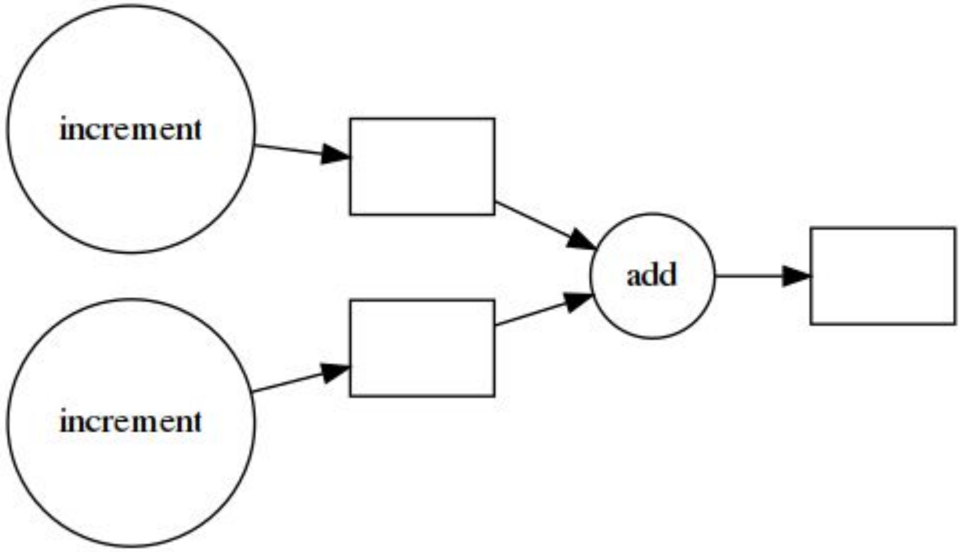
Example of computation DAG

Let's suppose to have a program that:

- Takes two integer and increment them
- After the incrementation, it calculates the sum of the two new numbers

```
x = delayed(increment)(1)
y = delayed(increment)(2)
z = delayed(add)(x, y)
```

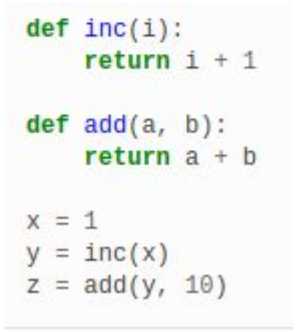
Dask generates the computation DAG from the code.
Each node of this graph is executed by a computing node of a cluster



Example of computation DAG

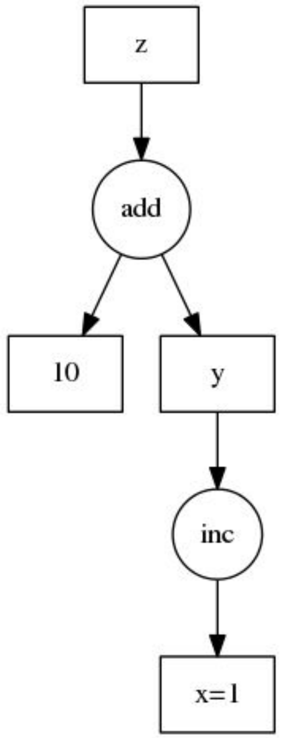
Let's suppose to have a program that:

- Takes two integer
- Increment one of them
- Assing the incremented number to a variable
- Sum the final numbers

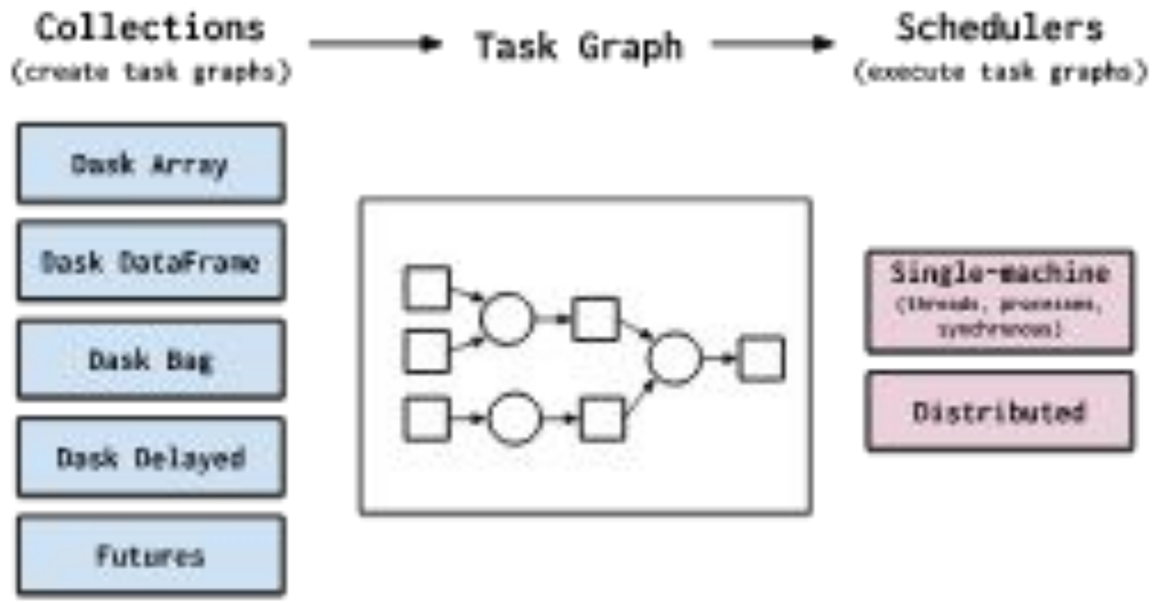


```
def inc(i):
    return i + 1

def add(a, b):
    return a + b

x = 1
y = inc(x)
z = add(y, 10)
```

DAG:



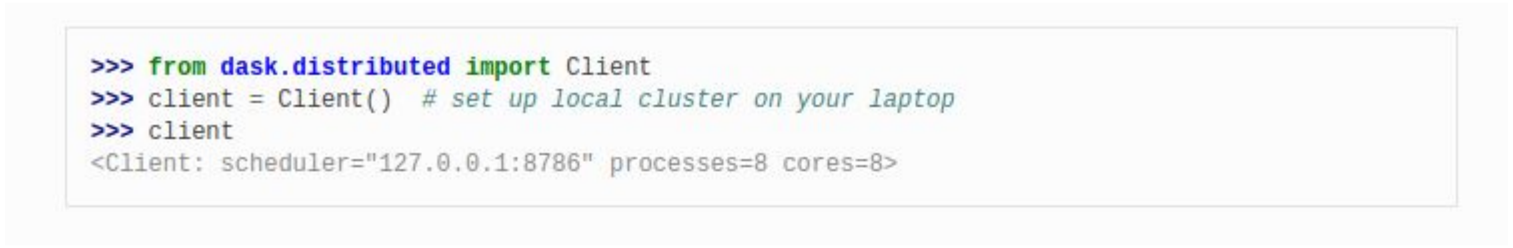
Summarizing



How to create a simple Cluster?

Dask allows to build “Local cluster”
Your computer becomes a cluster, where each core of your CPU represents a computing nodes of the cluster (dask-worker), while a single core is used as head/master node (dask-scheduler)

You can do this directly on you python code.

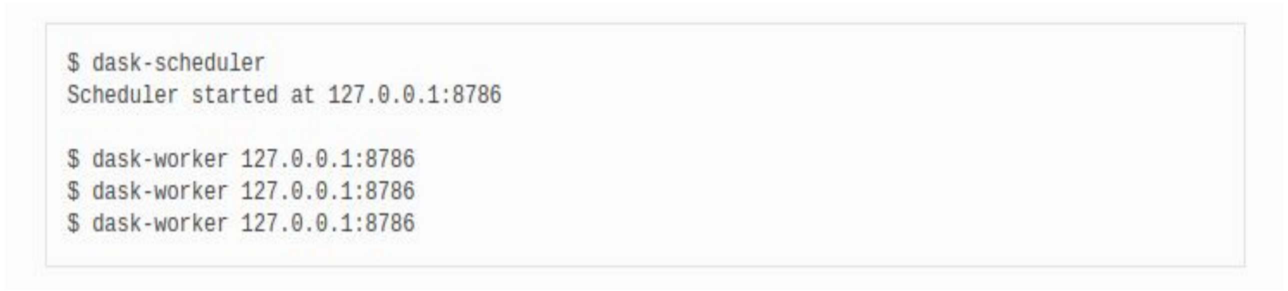


```
>>> from dask.distributed import Client
>>> client = Client()  # set up local cluster on your laptop
>>> client
<Client: scheduler="127.0.0.1:8786" processes=8 cores=8>
```

How to create a simple multi-machine Cluster?

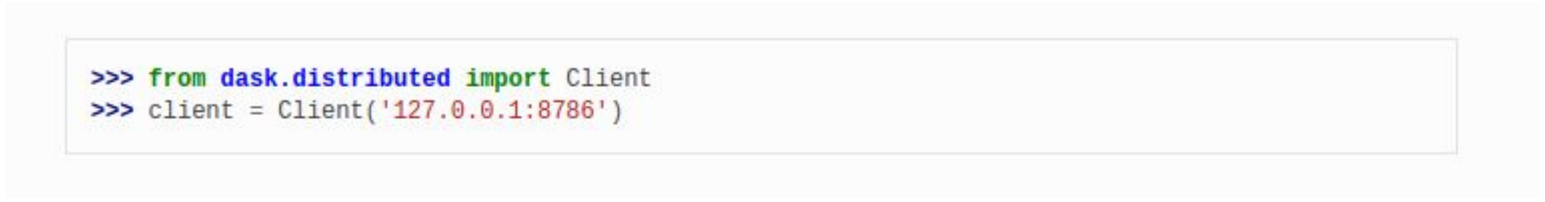
At the same time Dask made simple to create a cluster by using different resources:

- command **dask-scheduler** must be typed on the scheduler machine (like your local laptop)
- command **dask-worker ip:port** must be typed on each computing node. the **ip** variable, is the ip address of scheduler, while the variable **port** is the communication port between scheduler and workers



```
$ dask-scheduler
Scheduler started at 127.0.0.1:8786

$ dask-worker 127.0.0.1:8786
$ dask-worker 127.0.0.1:8786
$ dask-worker 127.0.0.1:8786
```

Then on your python code just type these two instructions. Through the usage of variable **client** , it's now possible to distribute some tasks

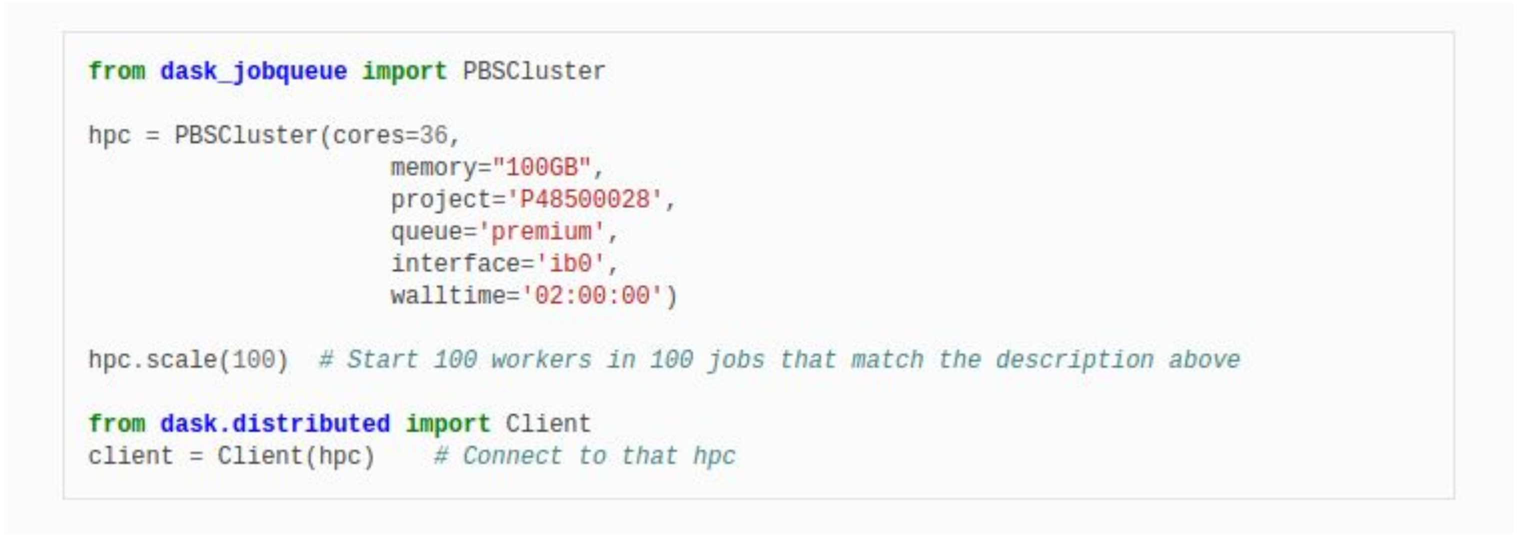


```
>>> from dask.distributed import Client
>>> client = Client('127.0.0.1:8786')
```

How to create a simple HPC?

Dask made simple also to work with HPC systems directly in Python, where you have to define:

- CPU core requirements (36)
- The RAM memory requirements (100GB)
- Network communication interface (ib0)
- The execution time of the required resource (2h)
- Start at least the number of compute nodes that match the requirements (100 nodes)



```python
from dask_jobqueue import PBSCluster

hpc = PBSCluster(cores=36,
                     memory="100GB",
                     project='P48500028',
                     queue='premium',
                     interface='ib0',
                     walltime='02:00:00')

hpc.scale(100)  # Start 100 workers in 100 jobs that match the description above

from dask.distributed import Client
client = Client(hpc)    # Connect to that hpc
```