High Performance Computing with Python

Summer School in Dynamic Structural Econometrics
Lausanne, August 21-26, 2023

Rafael Sarmiento
ETHZürich / CSCS

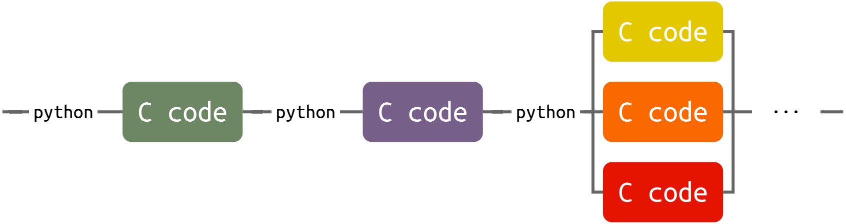
ETH zürich CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

Python

- Most of its operations can be customized
- It’s fairly easy to glue it to other languages like C and Fortran

Python

- Most of its operations can be customized
- It’s fairly easy to glue it to other languages like C and Fortran



Python in HPC

- **NumPy-like operations with multidimensional arrays**
  - NumPy, CuPy, PyTorch, TensorFlow, Jax, ...
  - Many packages have reimplemented NumPy's API providing support on platforms like GPUs and TPUs
- **Compiled code**
  - Numba, Cython, Taichi Lang but also PyTorch and TensorFlow
  - Support for multiple platforms
  - Interfaces to languages like Fortran90 and C/C++ (pybind11, CFFI, F2PY, ...)
- **Scaling workflows in clusters**
  - dask / dask.distributed, PySpark, IPyParallel, MPI4Py, CuNumerics, ...
  - Scalable reimplementations of NumPy

NumPy-like operations with multidimensional arrays

ETH zürich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

NumPy is a Python library that adds support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- `numpy.ndarray`: a powerful N-dimensional array object
- Sophisticated functions often written in C
- Linear algebra, Fourier transform, and random number capabilities
- Tools for easy binding to Fortran code (F2PY)
- Compatibility with C
- Many libraries implement the NumPy API, such as Dask and CuPy for graph and GPU computing respectively [NEP 35]

CuPy is an open-source array library accelerated with NVIDIA CUDA. It provides GPU accelerated computing with Python

- CuPy uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT and NCCL
- CuPy's interface is highly compatible with NumPy: in most cases it can be used as a drop-in replacement
- It compiles a kernel code optimized for the shapes and dtypes of given arguments, sends it to the GPU device, and executes the kernel

The **SciPy** library provides many user-friendly and efficient numerical routines for operations such as numerical integration, interpolation, optimization, linear algebra and statistics. SciPy builds on the `numpy.ndarray` and expands the set of mathematical functions included in NumPy

Vectorization

- Use operations over the whole array instead of over single elements.

```
z = x + y            # x = np.array([...])
                     # y = np.array([...])
```

Vectorization

- Use operations over the whole array instead of over single elements.

```
z = x + y          # x = np.array([...])
                   # y = np.array([...])
```

- When working with arrays, use *ufuncs* and general NumPy's functions.

```
y = np.exp(x)      # x = np.array([...])
z = x @ y          # equivalent to np.matmul(x, y)
```

Vectorization

- Use operations over the whole array instead of over single elements.

```
z = x + y               # x = np.array([...])
                        # y = np.array([...])
```

- When working with arrays, use *ufuncs* and general NumPy's functions.

```
y = np.exp(x)           # x = np.array([...])
z = x @ y               # equivalent to np.matmul(x, y)
```

- Adapt your solutions to use the two points above.

Euclidean distance matrix

$$d_e\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum(x_{1i}-y_{1i})^2 & \sum(x_{1i}-y_{2i})^2 & \dots & \sum(x_{1i}-y_{ni})^2 \\ \sum(x_{2i}-y_{1i})^2 & \sum(x_{2i}-y_{2i})^2 & \dots & \sum(x_{2i}-y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum(x_{ni}-y_{1i})^2 & \sum(x_{ni}-y_{2i})^2 & \dots & \sum(x_{ni}-y_{ni})^2 \end{bmatrix}$$

Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

Euclidean distance matrix

$$d_e\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum(x_{1i}-y_{1i})^2 & \sum(x_{1i}-y_{2i})^2 & \dots & \sum(x_{1i}-y_{ni})^2 \\ \sum(x_{2i}-y_{1i})^2 & \sum(x_{2i}-y_{2i})^2 & \dots & \sum(x_{2i}-y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum(x_{ni}-y_{1i})^2 & \sum(x_{ni}-y_{2i})^2 & \dots & \sum(x_{ni}-y_{ni})^2 \end{bmatrix}$$

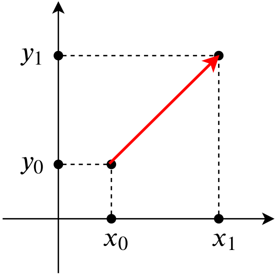
Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

Euclidean distance matrix

$$d_e\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

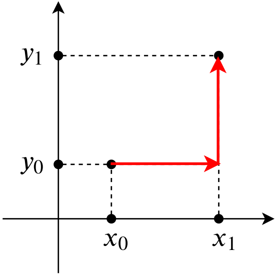
Euclidean distance matrix



$$d_e\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \cdots & \cdots & \cdots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) =$$

$$\begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \cdots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \cdots & \sum (x_{2i} - y_{ni})^2 \\ \cdots & \cdots & \cdots & \cdots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \cdots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

Cityblock distance matrix



$$d_{\text{cb}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) =$$

$$\begin{bmatrix} \sum |x_{1i} - y_{1i}| & \sum |x_{1i} - y_{2i}| & \dots & \sum |x_{1i} - y_{ni}| \\ \sum |x_{2i} - y_{1i}| & \sum |x_{2i} - y_{2i}| & \dots & \sum |x_{2i} - y_{ni}| \\ \dots & \dots & \dots & \dots \\ \sum |x_{ni} - y_{1i}| & \sum |x_{ni} - y_{2i}| & \dots & \sum |x_{ni} - y_{ni}| \end{bmatrix}$$

```python
def euclidean_distance_matrix(x, y):
    num_samples = x.shape[0]
    dist_matrix = np.empty((num_samples,
                            num_samples))
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            diff = xi - yj
            dist_matrix[i][j] = diff @ diff

    return dist_matrix
```

```python
def euclidean_distance_matrix(x, y):
    num_samples = x.shape[0]
    dist_matrix = np.empty((num_samples,
                            num_samples))
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            diff = xi - yj
            dist_matrix[i][j] = diff @ diff

    return dist_matrix
```

- ✗ Use operations over the whole array instead of over single elements.
- ✓ When working with arrays, use ufuncs and general NumPy's functions.
- ✗ Adapt your solutions to use the two points above.

Implementation #1

```python
def euclidean_distance_matrix(x, y):
    diff = x[:, np.newaxis, :] - y[np.newaxis, :, :]

    return (diff * diff).sum(axis=2)
```

Implementation #2

$$\sum_k (x_{ik} - y_{jk})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

Implementation #2

$$\sum_k (x_{ik} - y_{jk})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \rightarrow$ `x @ y.T` : Matrix product of $\{\vec{x}\}$ and $\{\vec{y}\}$

Implementation #2

$$\sum_k (x_{ik} - y_{jk})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \longrightarrow$ `x @ y.T` : Matrix product of $\{\vec{x}\}$ and $\{\vec{y}\}$

$\vec{x}_i \cdot \vec{x}_i \longrightarrow$ `(x * x).sum(axis=1)` : A vector of elements $\sum_j x_{ij}x_{ij} \equiv \sum_j x_{ij}^2$

$\vec{y}_j \cdot \vec{y}_j \longrightarrow$ `(y * y).sum(axis=1)` : A vector of elements $\sum_j y_{ij}y_{ij} \equiv \sum_j y_{ij}^2$

Implementation #2

$$\sum_k (x_{ik} - y_{jk})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \rightarrow$ `x @ y.T`

$\vec{x}_i \cdot \vec{x}_i \rightarrow$ `(x * x).sum(axis=1)[:, np.newaxis]`

$\vec{y}_j \cdot \vec{y}_j \rightarrow$ `(y * y).sum(axis=1)[np.newaxis, :]`

Implementation #2

```python
def euclidean_distance_matrix(x, y):
    x2 = (x * x).sum(axis=1)[:, np.newaxis]
    y2 = (y * y).sum(axis=1)[np.newaxis, :]
    xy = x @ y.T

    return np.abs(x2 + y2 - 2 * xy)
```

Cityblock distance matrix

$$\sum_k |x_{ik} - y_{jk}|$$

The trick we used for the Euclidean distance matrix doesn't work here!

Compiled code

Numba is an open source just-in-time (JIT) compiler that translates a subset of Python and NumPy code into fast machine code.

- Translation of python functions to machine code at runtime using the LLVM compiler library
- Designed to be used with NumPy arrays
- Options to parallelize code for CPUs and GPUs and automatic SIMD Vectorization
- Support for both NVIDIA's CUDA and AMD's ROCm driver allowing to write parallel GPU code from Python.

```python
def reduce(x):
    x_sum = 0.0
    for i in range(x.shape[0]):
        x_sum += x[i]

    return x_sum
```

```python
import numba

@numba.jit(nopython=True)
def reduce(x):
    x_sum = 0.0
    for i in range(x.shape[0]):
        x_sum += x[i]

    return x_sum
```

Scaling workflows in clusters

dask is a flexible library for parallel computing in Python. It provides dynamic task scheduling optimized for computation as well as big data collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy and Pandas to larger-than-memory or distributed environments.

- `dask.delayed` can be used to parallelize custom algorithms by creating computational graphs.

```
# regular code           |   # with dask
x = func1(<args>)        |   x = dask.delayed(func1)(<args>)
y = func2(<args>)        |   y = dask.delayed(func2)(<args>)
z = func3(x, y)          |   z = dask.delayed(func3)(x, y)
                         |   #       x         y
                         |   #        \       /
                         |   #       func1  func2
                         |   #         \     /
                         |   #          func3
                         |   #            |
                         |   #            z
                         |   z.compute(scheduler='threads')
```

- `dask.delayed` can be used to parallelize custom algorithms by creating computational graphs.

```
list_delayed = [dask.delayed(func1)(<args>),
                dask.delayed(func2)(<args>),
                dask.delayed(func3)(<args>)]

dask.compute(*list_delayed, scheduler='threads')
```

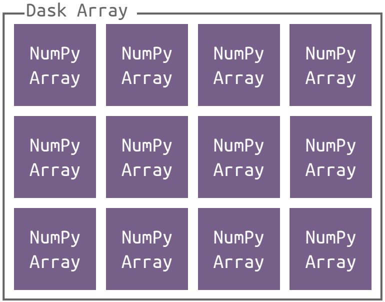
- `dask.array` implements a subset of the NumPy array interface using blocked algorithms, cutting up the large array into chunks of small arrays.
- `dask.bag` parallelizes computations across a large collection of generic Python objects.
- `dask.dataframe` is a large parallel DataFrame composed of many smaller Pandas DataFrames which may live on disk for larger-than-memory computing on a single machine or a cluster.

dask array

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

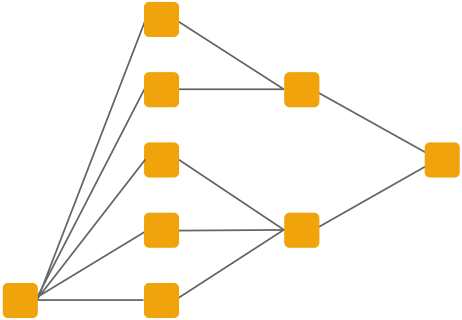
dask array

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



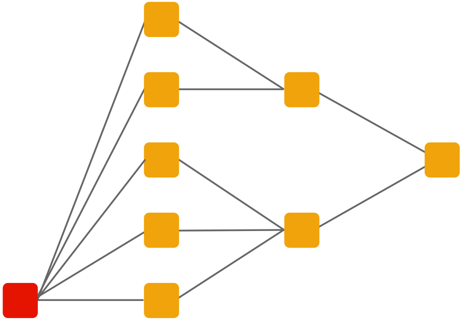
- A dask array consists of many NumPy arrays arranged into a grid
- Those NumPy arrays may live on memory, disk or remote machines
- `dask.array` implements many of the numpy functions but in block-wise fashion and are executed through a graph.
- For equal sizes, operations on dask arrays are in general slower than the corresponding NumPy ones.

`dask.array` graph



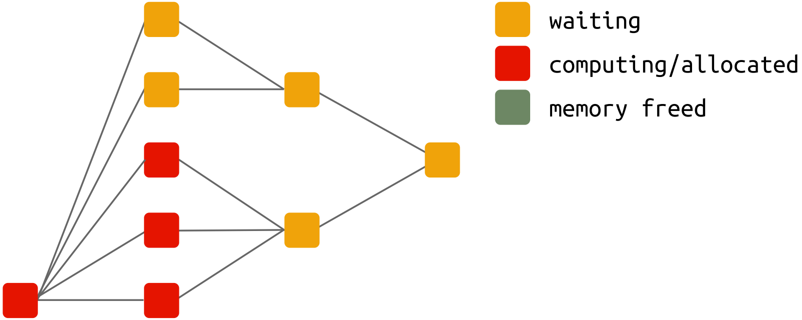
- waiting
- computing/allocated
- memory freed

`dask.array` graph

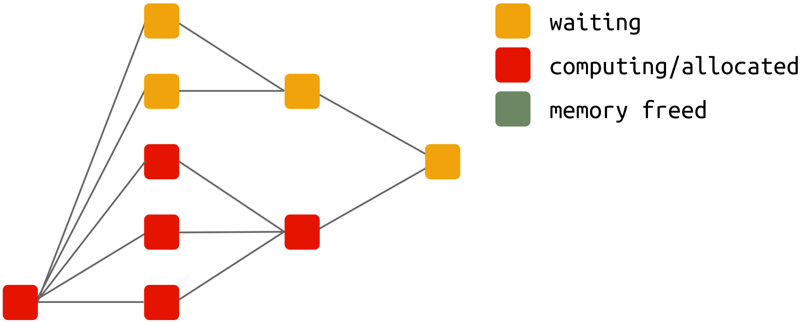


- waiting
- computing/allocated
- memory freed

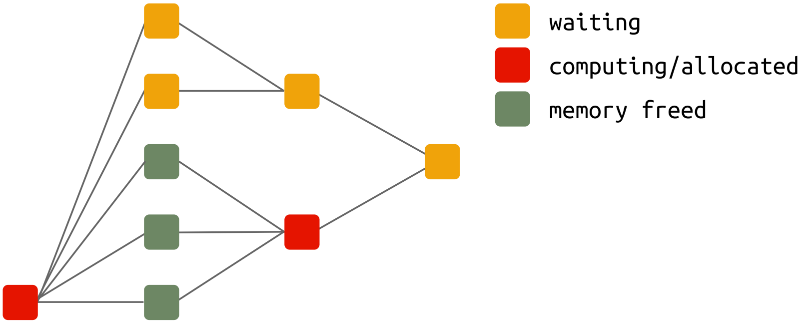
`dask.array` graph



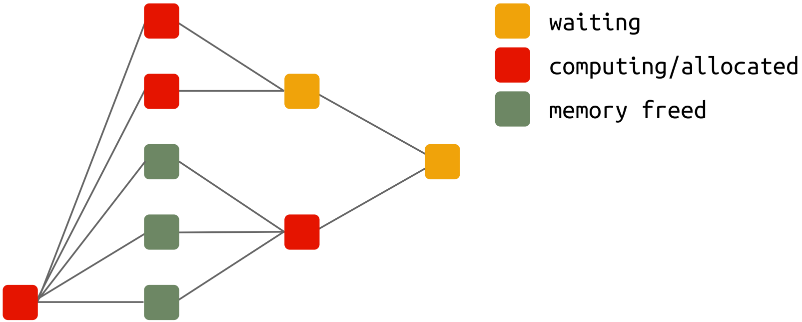
`dask.array` graph



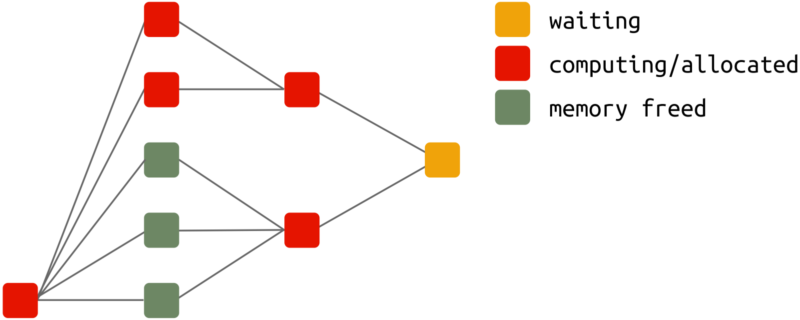
`dask.array` graph



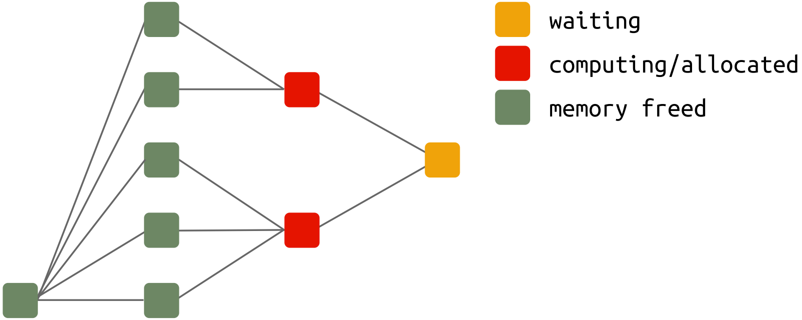
`dask.array` graph



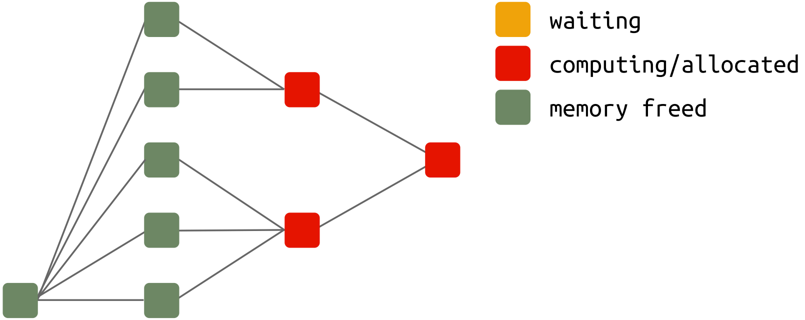
`dask.array` graph



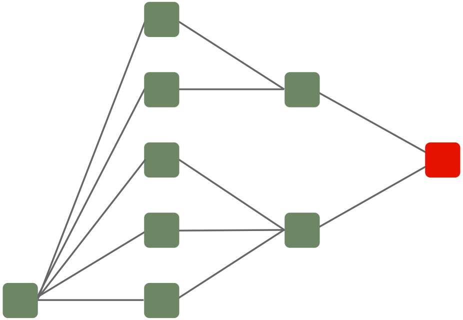
`dask.array` graph



`dask.array` graph



`dask.array` graph



waiting

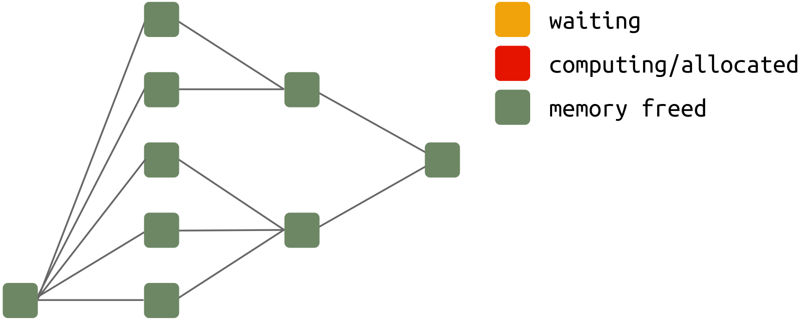
computing/allocated

memory freed

ETH zürich

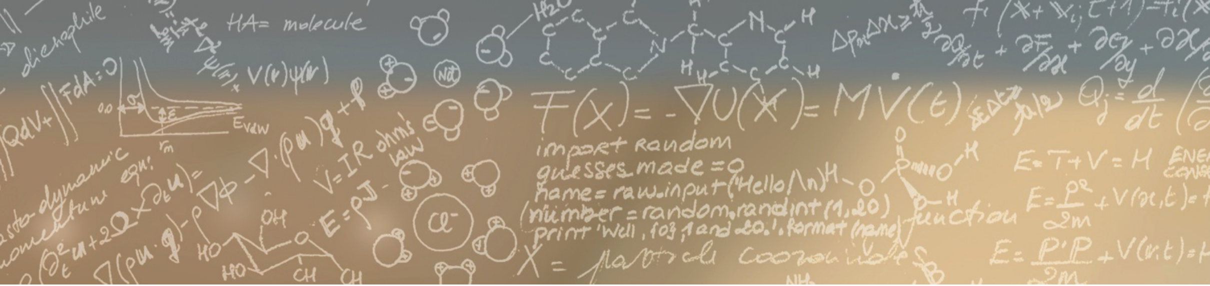
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

`dask.array` graph



More content

- Material for the course “HPC with Python” at CSCS
  https://github.com/eth-cscs/PythonHPC



Thank you for your attention!