OPEN DATA SCIENCE EUROPE WORKSHOP

High performance computing in python

Sept 7, 2021: 11:00 - 12:30

Leandro Parente

leandro.parente@opengeohub.org

https://opengeohub.org

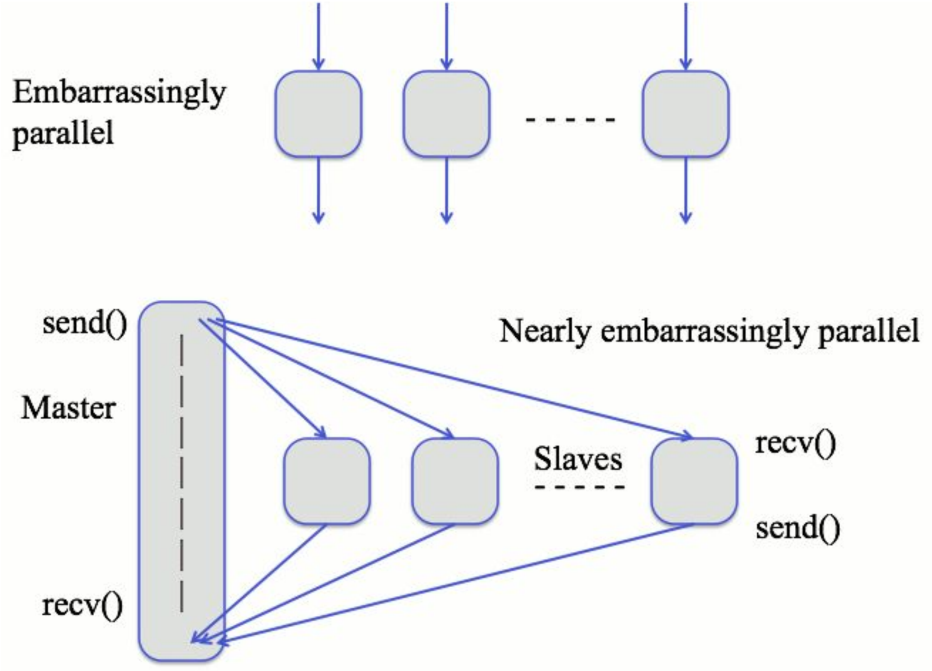
Co-financed by the Connecting Europe
Facility of the European Union

Introduction to ODSE datasets in Python - Outline

- Embarrassingly parallel problems
- Possibilities to optimize a raster processing workflow
- BLAS and LAPACK implementations
- Optimizing a temporal array reduction and a numeric operations
- Production workflow using tile processing

Embarrassingly parallel problems

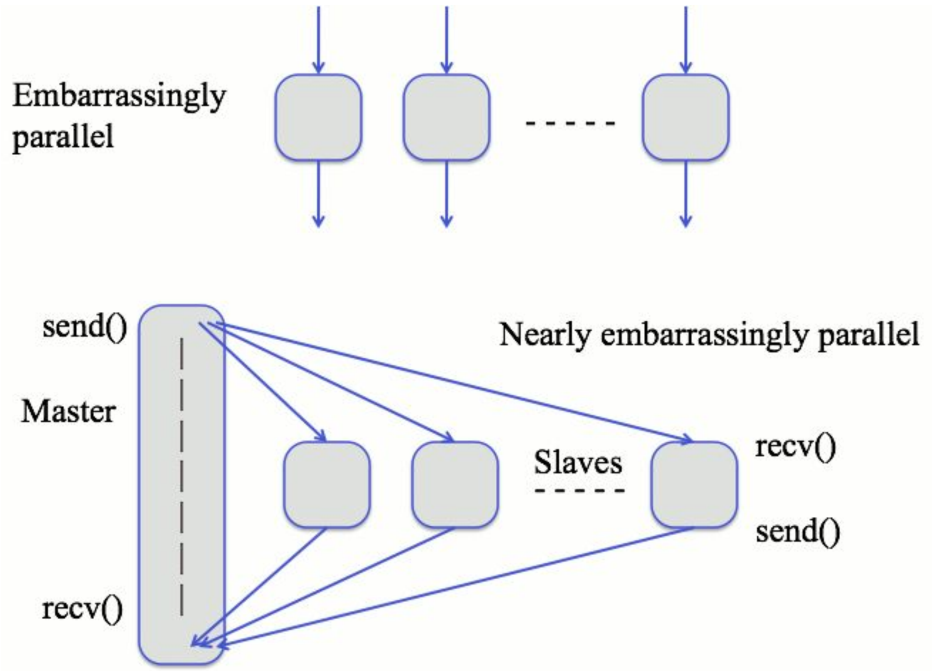
- Can be divided into completely **independent** parts,
- Requires none or very little communication,
- **Nearly embarrassingly parallel** is an embarrassingly parallel computation that requires initial data to be distributed and final results to be collected in some way



Source: [Embarrassingly Parallel Computations]() & [Embarrassingly Parallel Algorithms]()

Embarrassingly parallel problems

- Can be divided into completely **independent** parts,
- Requires none or very little communication,
- **Nearly embarrassingly parallel** is an embarrassingly parallel computation that requires initial data to be distributed and final results to be collected in some way

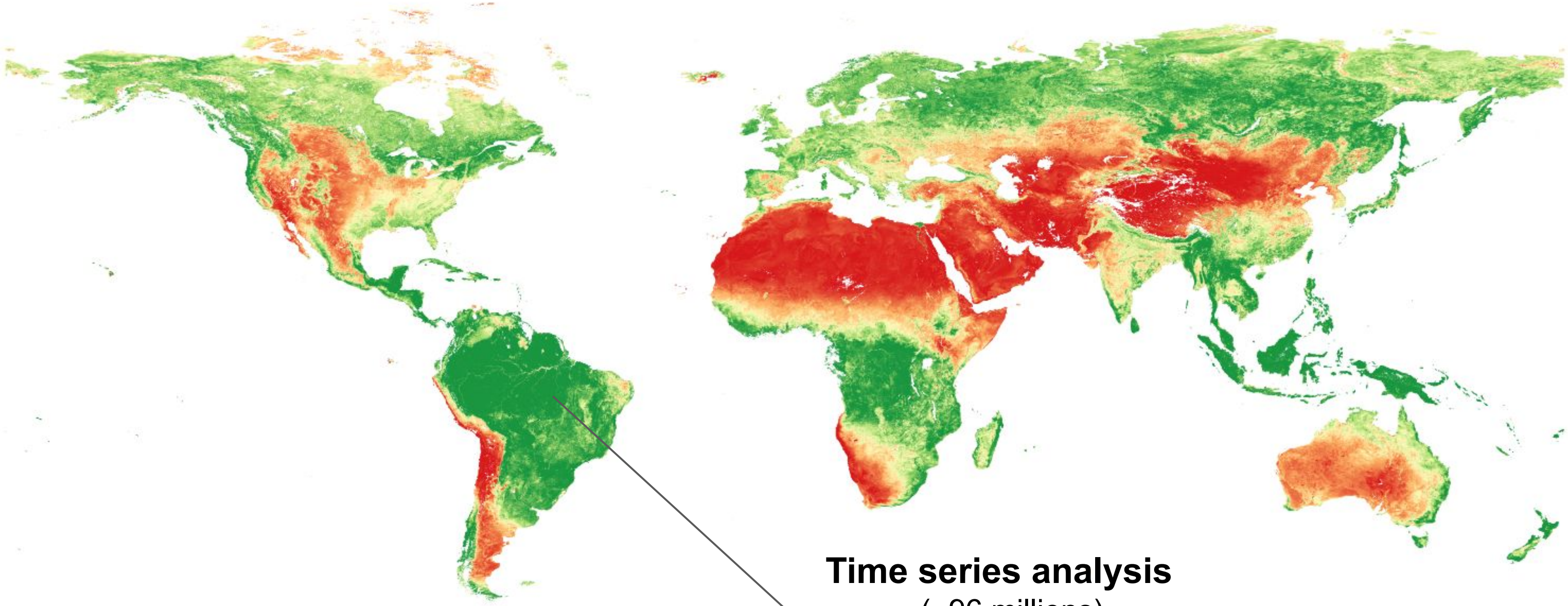


Raster processing

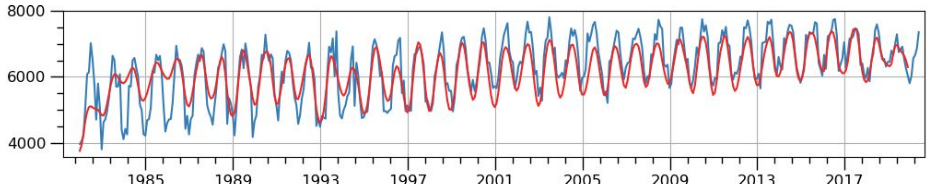
Source: [Embarrassingly Parallel Computations]() & [Embarrassingly Parallel Algorithms]()

OpenDataScience

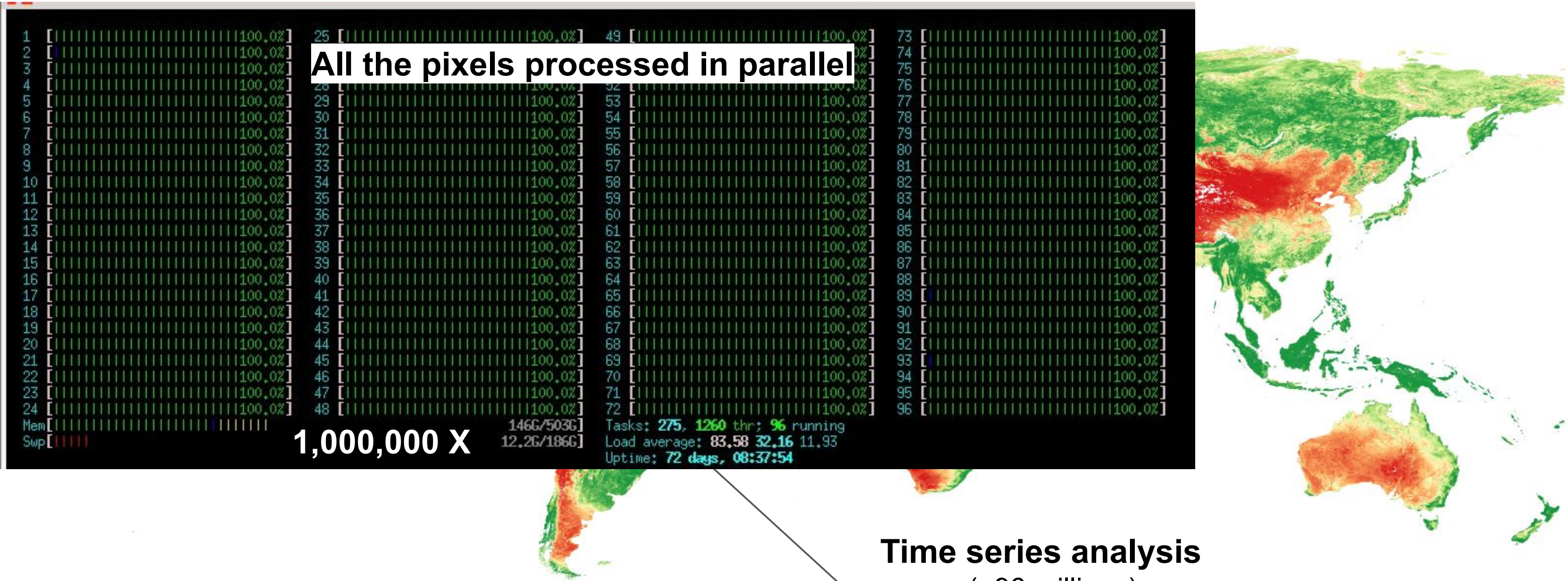
Embarrassingly parallel problems



Time series analysis
(~96 millions)



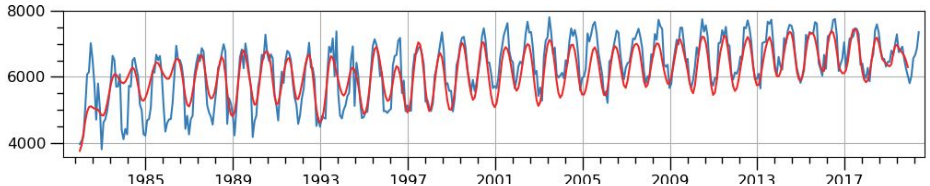
Embarrassingly parallel problems



All the pixels processed in parallel

1,000,000 X

Time series analysis
(~96 millions)



Possibilities to optimize a raster processing workflow

- Increase the number of CPU cores
- Improve data transfer speed
- Improve the processing code (new algorithms/functions):

Possibilities to optimize a raster processing workflow

- Increase the number of CPU cores
- Improve data transfer speed
- Improve the processing code (new algorithms/functions):
  - Drop-in replacement
  - New code implementation

BLAS and LAPACK implementations

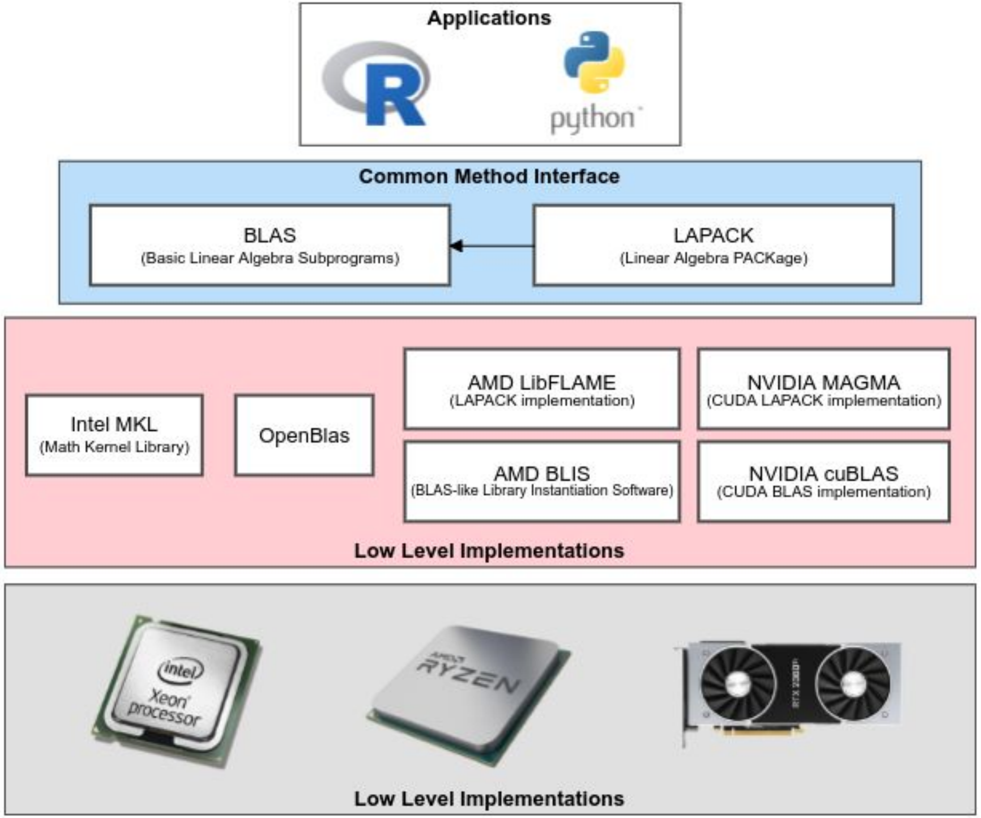
[BLAS (Basic Linear Algebra Subprograms)]() is a C library to provide a set of routines for basic vector and matrix operations

[LAPACK (Linear Algebra PAckage)]() Fortran 90 library to solve linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, singular value problems and the associated matrix factorization

Level 1 BLAS

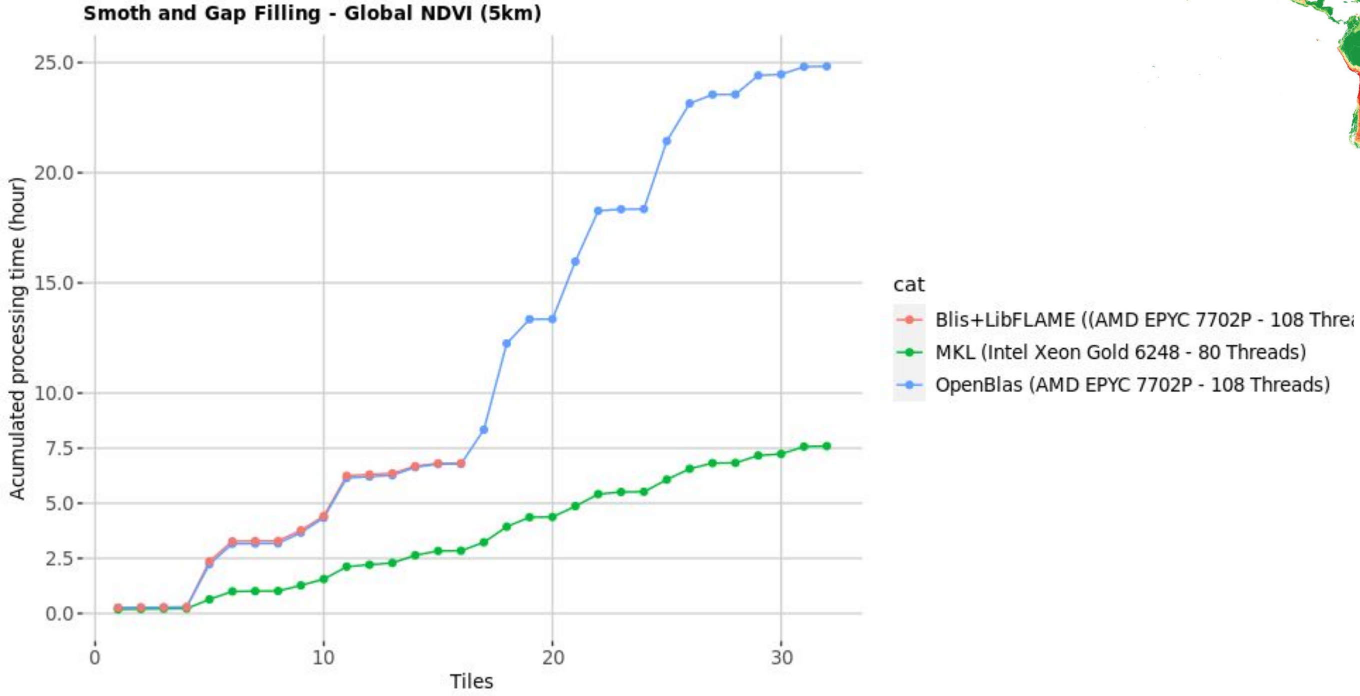
		dim	scalar	vector	vector	scalars	5-element array		prefixes
SUBROUTINE	xROTG (					A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE	xROTMG(					D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE	xROT (	N,		X, INCX,	Y, INCY,	C, S)		Apply plane rotation	S, D
SUBROUTINE	xROTM (	N,		X, INCX,	Y, INCY,		PARAM)	Apply modified plane rotation	S, D
SUBROUTINE	xSWAP (	N,		X, INCX,	Y, INCY)			$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE	xSCAL (	N,	ALPHA,	X, INCX)				$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE	xCOPY (	N,		X, INCX,	Y, INCY)			$y \leftarrow x$	S, D, C, Z
SUBROUTINE	xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)			$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION	xDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	S, D, DS
FUNCTION	xDOTU (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	C, Z
FUNCTION	xDOTC (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^H y$	C, Z
FUNCTION	xxDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION	xNRM2 (	N,		X, INCX)				$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
FUNCTION	xASUM (	N,		X, INCX)				$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
FUNCTION	IxAMAX(	N,		X, INCX)				$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

BLAS and LAPACK implementations

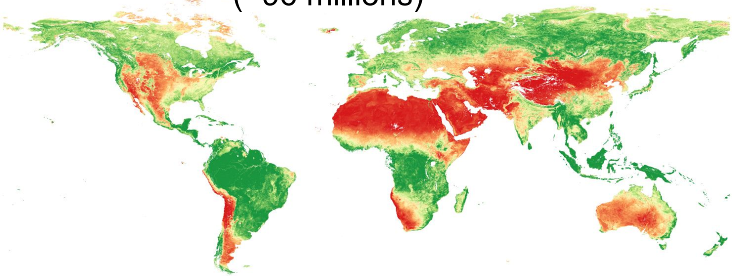


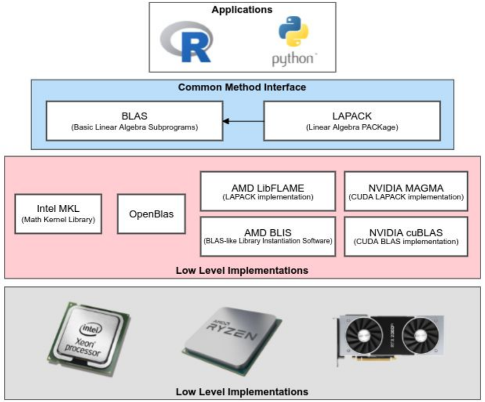
BLAS and LAPACK implementations

[MKL](#) is 3x faster then [OpenBlas](#)



Time series analysis
(~96 millions)





OpenDataScience

Optimizing a temporal array reduction and a numeric operations

https://gitlab.com/geoharmonizer_inea/odse-workshop-2021

06_HIGH_PERFORMANCE_COMPUTING.IPYNB

- High performance computing in Python
  - Array Reduction
  - Numeric Operation
  - Tiling processing

06_high_performance_c

Code

Python 3 (ipykernel)

High performance computing in Python

In this tutorial, you will learn how to use different Python libraries, compatible with Numpy, to reduce the computational time for multidimensional array operation. You will also check how to use a tiling strategy to manage raster datasets that does not fit in memory. At the end, you will combine these approaches in a processing workflow highly optimized for a single tile (chunk of raster data), but also able to process multiple tiles in parallel using available the CPU cores.

Additionally to Numpy, the follow libraries/modules will be used during the tutorial:

- Bootleneck: Collection of Numpy fast functions written in C, which are able to manage NaN values,
- NumExpr: Numeric expression evaluator memory optimized,
- Numba: Just-in-time Python compiler able to translate user defined numerical algorithms in C or FORTRAN,
- eumap.parallel: eumap implementation for multi-core parallelization strategies, written purely in Python.

First, let's import some common python modules and libraries

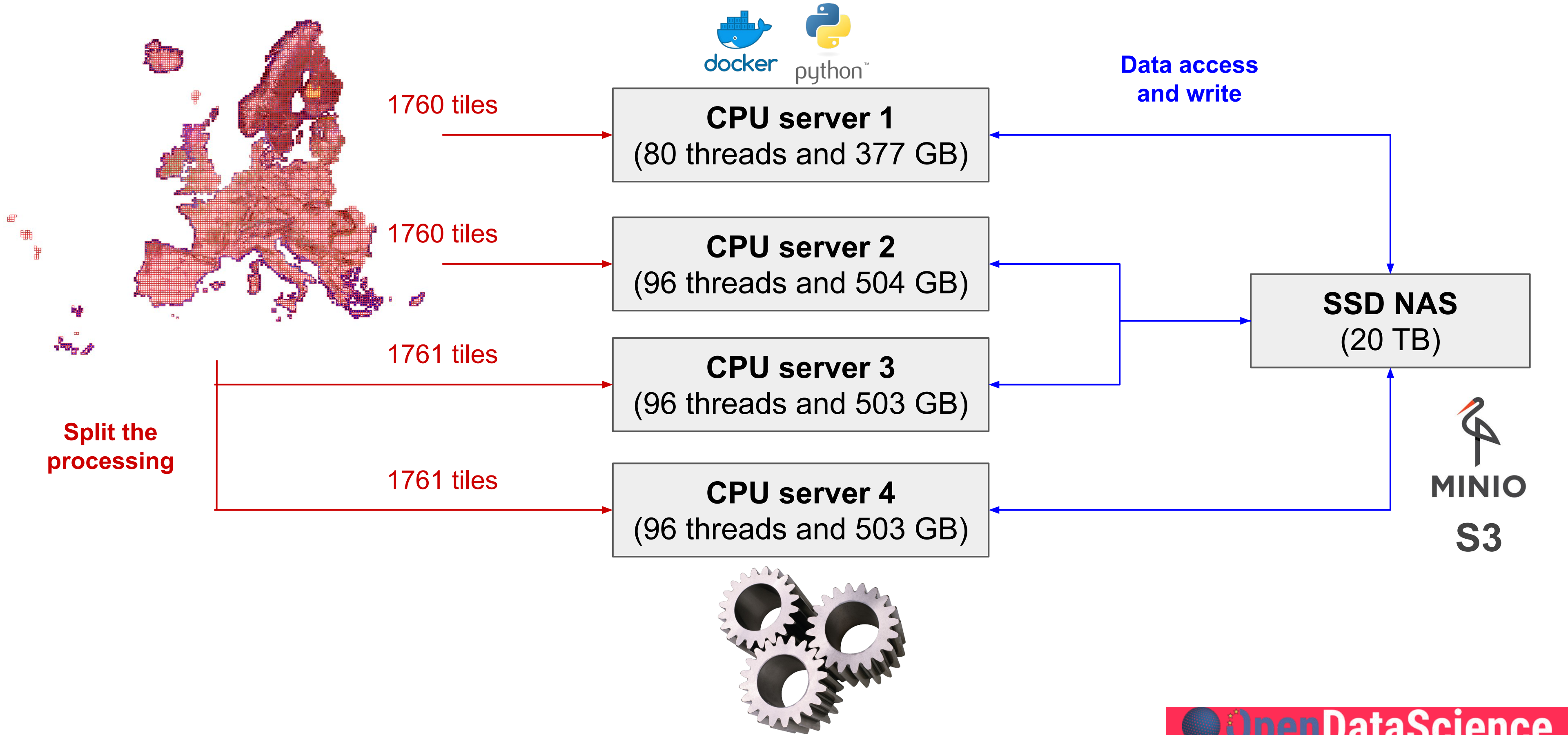
[15]:
```
from pathlib import Path
import numpy as np

from eumap.misc import find_files
from eumap.raster import read_rasters, save_rasters
from eumap import plotter
```

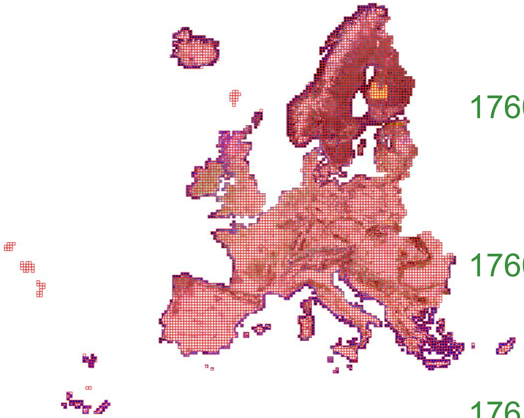
... and setup our `raster_dir`:

[16]:
```
ODSE_DATA_DIR = Path.home().joinpath('ODSE_workdir').joinpath('data')
```

Production workflow using tile processing



Production workflow using tile processing



docker python™

Data access

1760 tiles

CPU server 1
(80 threads and 377 GB)

1760 tiles

CPU server 2
(96 threads and 504 GB)

SSD NAS
(20 TB)

1761 tiles

CPU server 3
(96 threads and 503 GB)

Aggregate the results

1761 tiles

CPU server 4
(96 threads and 503 GB)

MINIO

S3

GDAL

Cloud optimized

tiff generation

gdalbuildvrt
gdal_translate

OpenDataScience