Joint Research Centre (JRC)

European Commission

Advanced geo-spatial data analysis with Jupyter

Davide DE MARCHI, Armin BURGER, Paul HASENOHR, Pierre SOILLE

European Commission, Joint Research Centre
Directorate I Competences, Unit I.3 Text and Data Mining
Big Data Analytics Project

Contacts: **armin.burger@ec.europa.eu**
pierre.soille@ec.europa.eu

Joint Research Centre

CS3 2019 – Cloud Storage Synchronization and Sharing Services
29/01/2019, Rome

DG Joint Research Centre

Established in **1957**

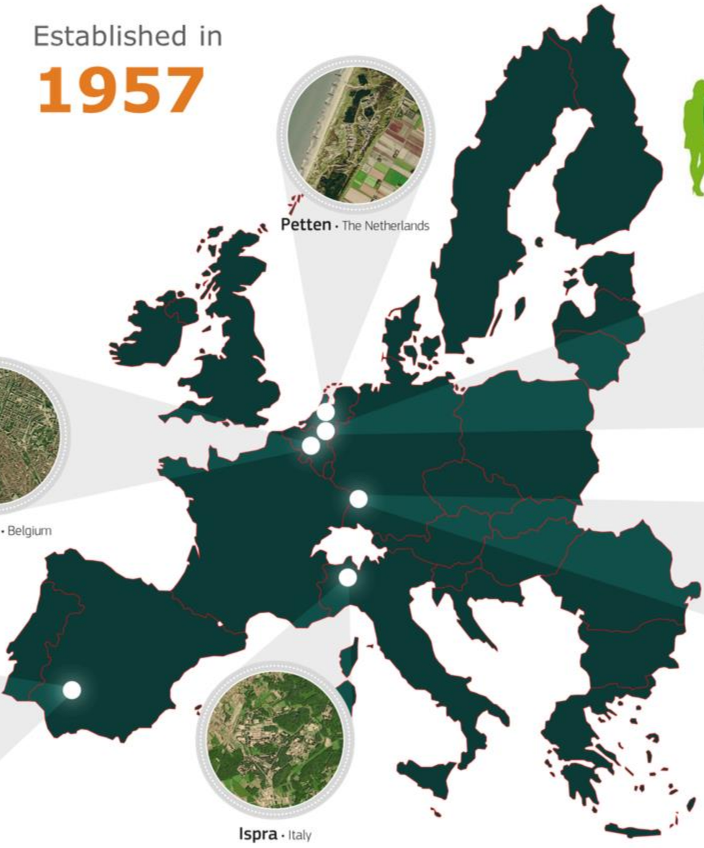
10 Directorates

6 Locations 5 Member States

3000 staff
Almost 75% are scientists and researchers.

>1000 Publications per year

42 Large scale facilities



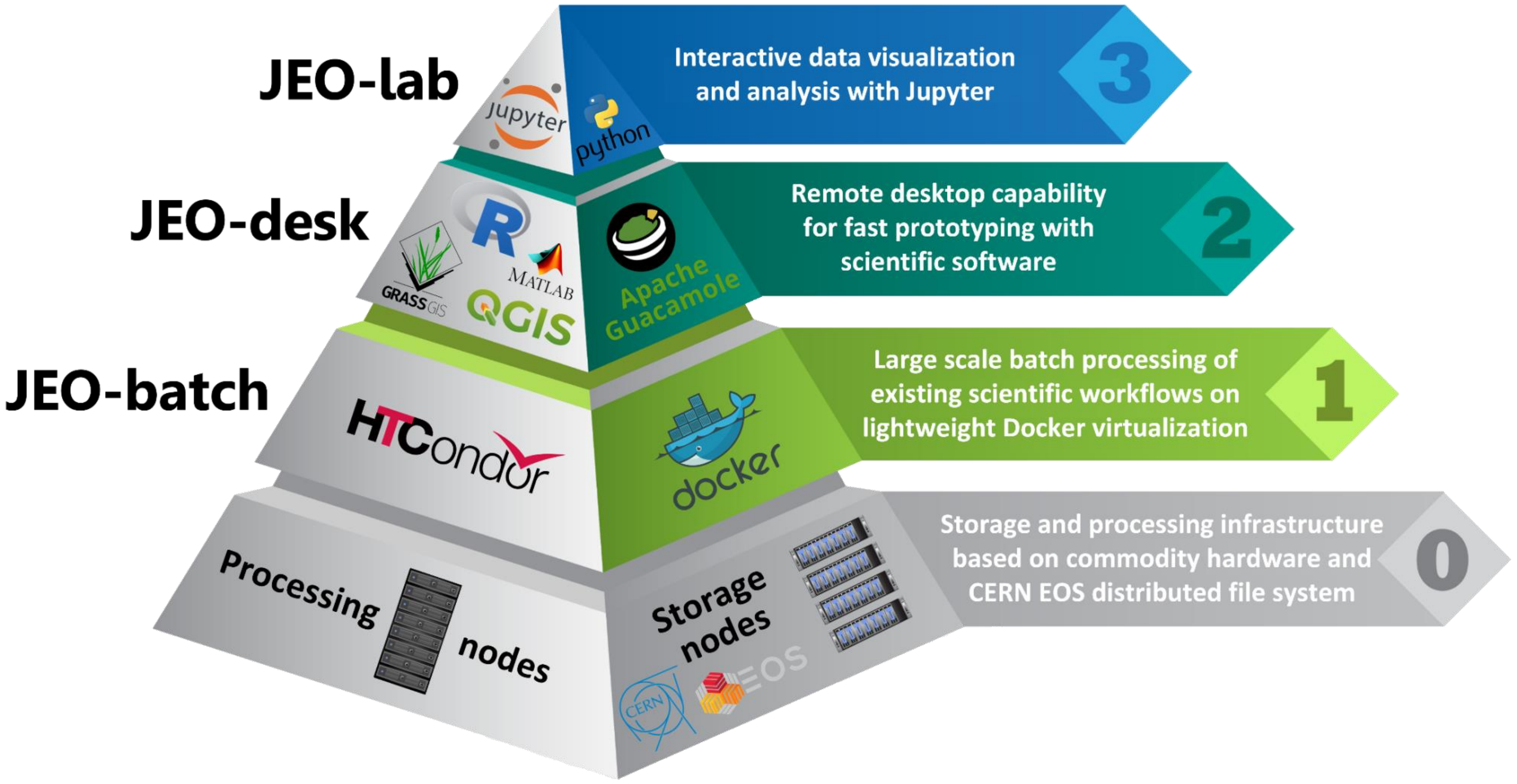
JEODPP: JRC Earth Observation Data and Processing Platform

Versatile platform bringing the users to the data and allowing for:

- Running large scale **batch processing** of existing scientific workflows thanks to lightweight virtualisation based on Docker
- **Remote desktop** capability for fast prototyping in legacy environments
- **Interactive data visualization and analysis** with Jupyter notebooks

Provides a collaborative environment serving the needs of users with very different requirements and skills

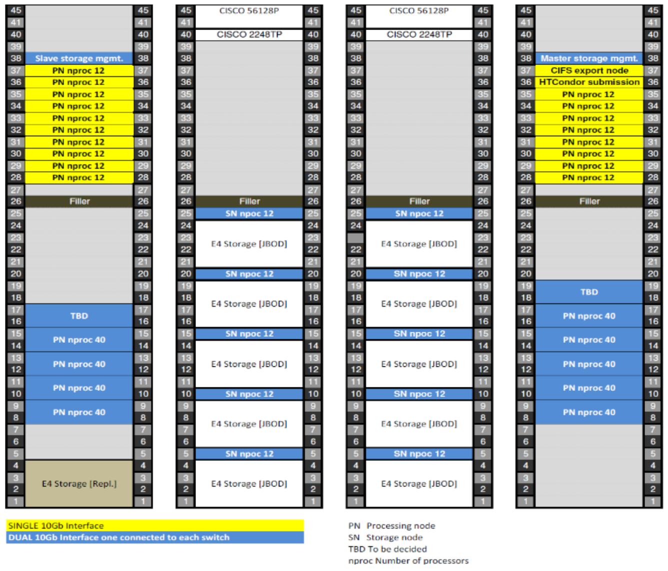
Conceptual representation


JEO-lab
jupyter
python
Interactive data visualization and analysis with Jupyter
3
JEO-desk
R
MATLAB
GRASS GIS
QGIS
Apache Guacamole
Remote desktop capability for fast prototyping with scientific software
2
JEO-batch
HTCondor
docker
Large scale batch processing of existing scientific workflows on lightweight Docker virtualization
1
Processing nodes
Storage nodes
CERN
EOS
Storage and processing infrastructure based on commodity hardware and CERN EOS distributed file system
0

Joint Research Centre

Europe Comm

Current status of JEODPP platform



Based on:

- **commodity** hardware
- **open-source** software stack

Storage:

- ***CERN EOS*** *distributed file system*
- *Currently 9 PB* ***net*** *capacity*

Processing servers:

- *1,500 cores over 35 nodes*
- *4 servers equipped with multi-GPUs and dedicated to Machine Learning processing with TensorFlow, Keras, …*

Main software stack

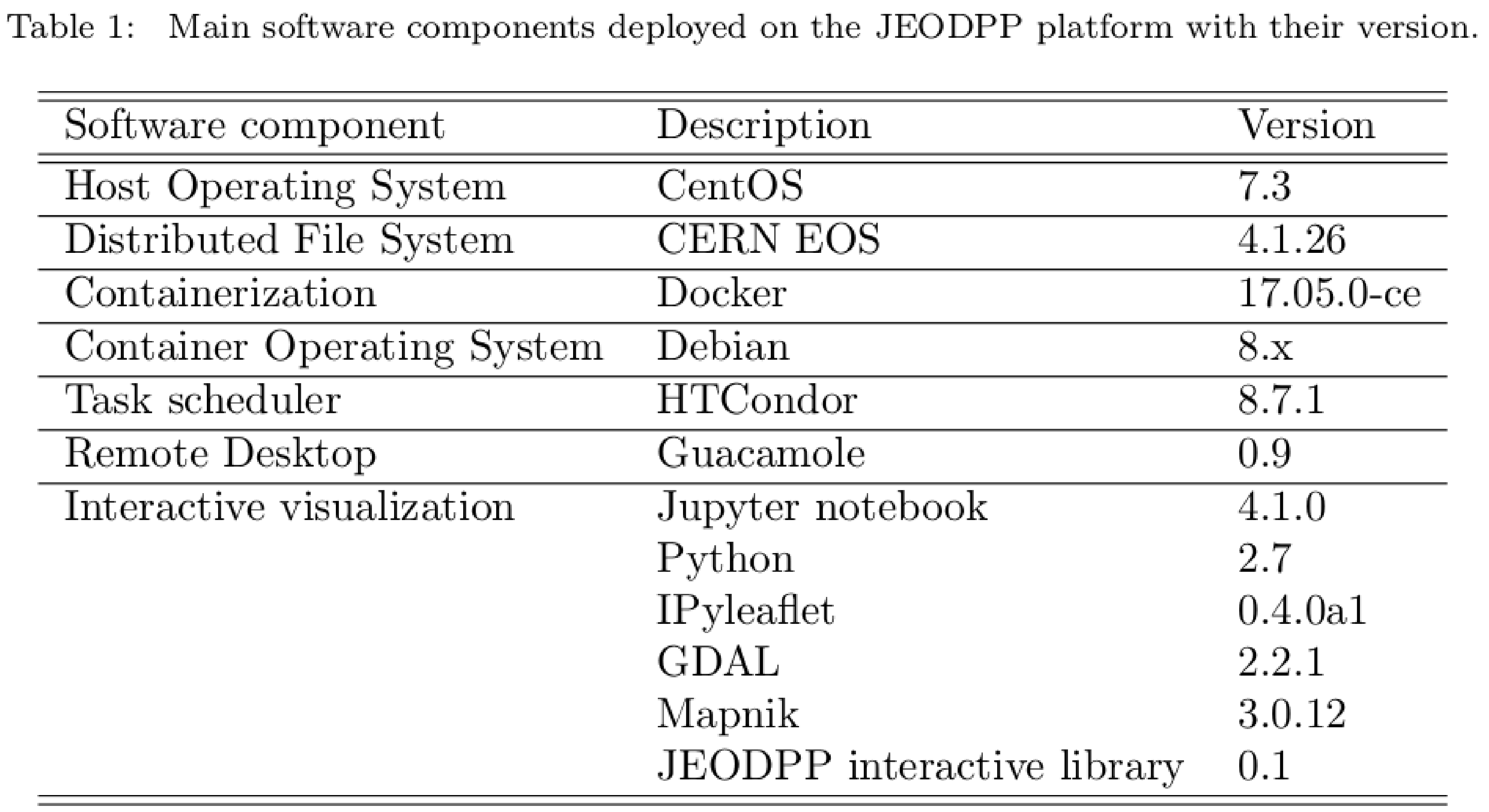


Table 1: Main software components deployed on the JEODPP platform with their version.

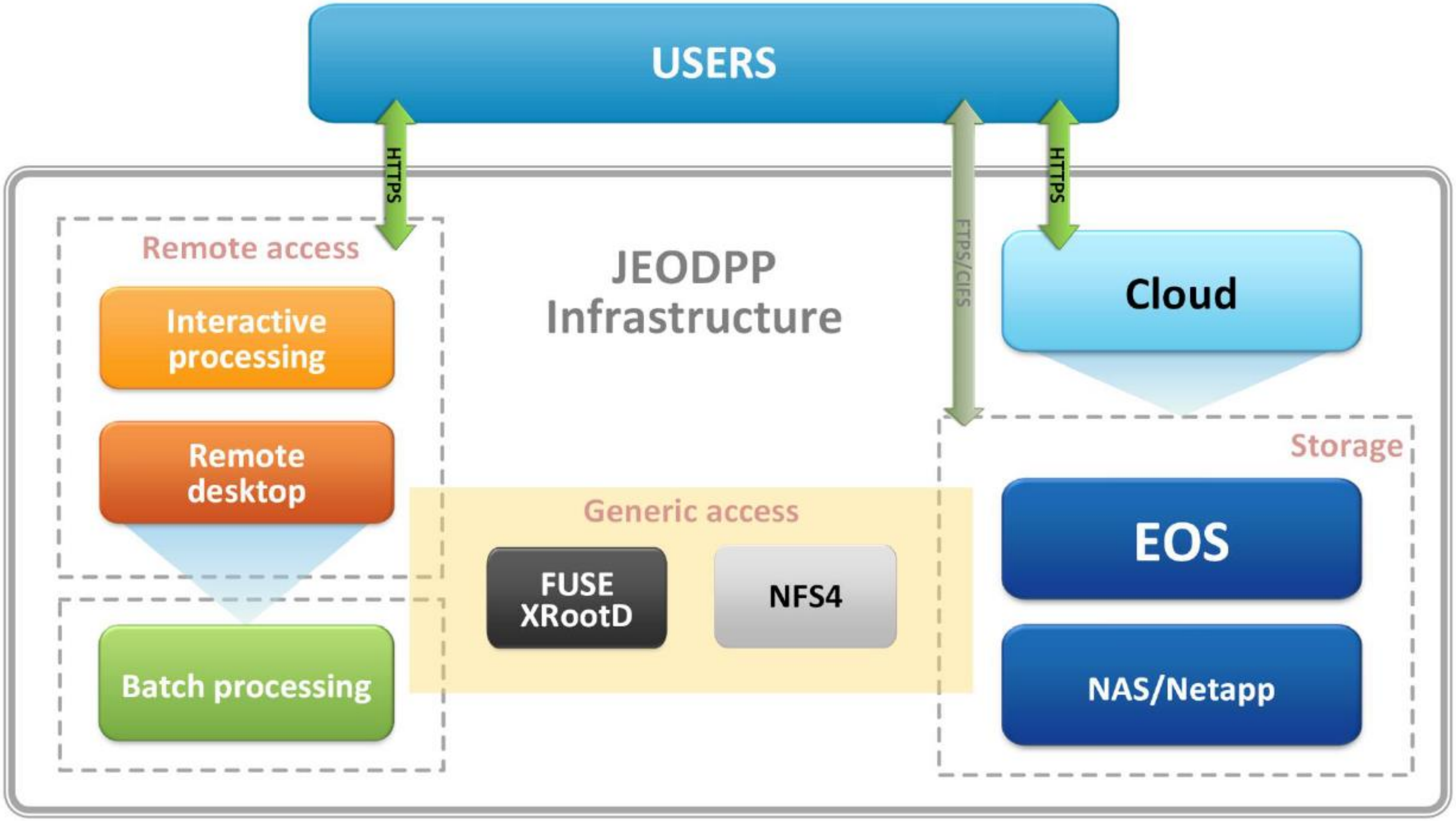
Software component	Description	Version
Host Operating System	CentOS	7.3
Distributed File System	CERN EOS	4.1.26
Containerization	Docker	17.05.0-ce
Container Operating System	Debian	8.x
Task scheduler	HTCondor	8.7.1
Remote Desktop	Guacamole	0.9
Interactive visualization	Jupyter notebook	4.1.0
	Python	2.7
	IPyleaflet	0.4.0a1
	GDAL	2.2.1
	Mapnik	3.0.12
	JEODPP interactive library	0.1

Source: Soille et al., Future Generation of Computer Systems, 2017 DOI: 1010.1016/j.future.2017.11.007 (in press)

Joint Research Centre

European Commission

Connecting storage and processing via cloud sharing services



JEODPP batch processing: JEO-batch

- Running large-scale data processing tasks in a cluster environment
- *Docker containers* for flexible management of processing environments
  - *Custom builds for different requirements*
  - *Facilitates upgrades of processing environment*
- Run through a workload manager:

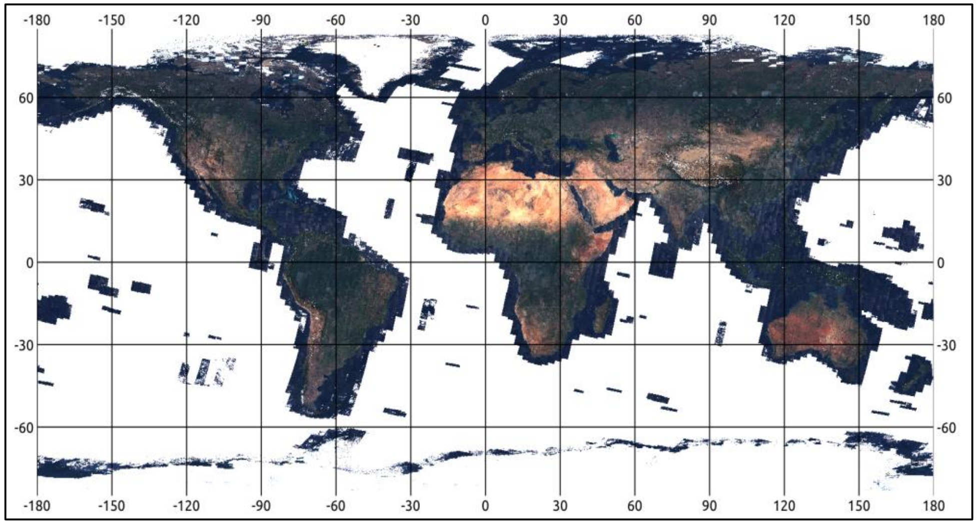
  HTCondor scheduler
- Extensive use for large scale processing/analysis

Joint Research Centre

European Commission

Optimizing Sentinel-2 image selection in a Big Data Context - Running in Docker universe

- "Optimal": 100% cloud free
- The selection was made from 2,128,556 optical remote sensing images.
- Each image was assigned to one core.
- No dependencies between jobs



Global S2 Quick look Mosaic doi:10.1080/20964471.2017.1407489

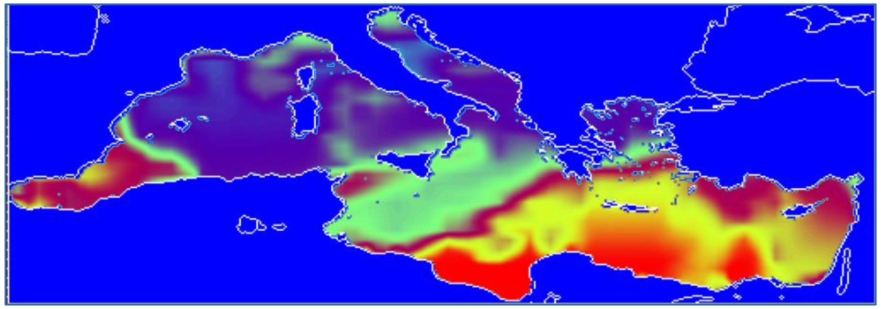
Joint Research Centre

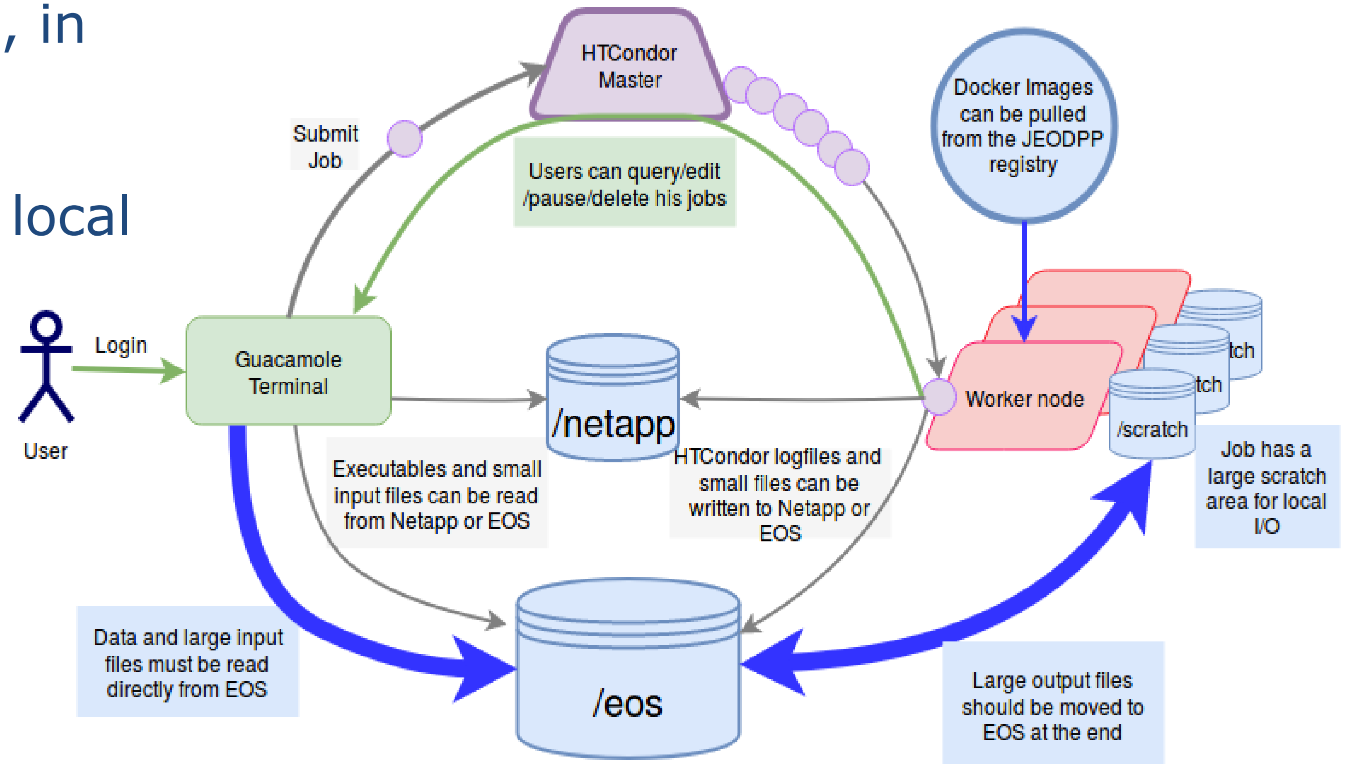
European Commission

Mediterranean Sea simulation 1958-2013
Running in Parallel universe + Docker Swarm

- 50 years simulation over the Mediterranean sea.
- CERN EOS is used for the main storage, in cooperation with NETAPP NFS and local scratch
- MPI application
- MESOS is used

Hydrodynamic and ecosystem simulations
http://mcc.jrc.ec.europa.eu





HTCondor Master
Docker Images can be pulled from the JEODPP registry
Submit Job
Users can query/edit /pause/delete his jobs
Login
User
Guacamole Terminal
/netapp
Worker node
/scratch
tch
tch
Job has a large scratch area for local I/O
Executables and small input files can be read from Netapp or EOS
HTCondor logfiles and small files can be written to Netapp or EOS
Data and large input files must be read directly from EOS
/eos
Large output files should be moved to EOS at the end

Dask on Kubernetes

From a Jupyter notebook, the Dask python package (https://dask.org) provides an interface to a Kubernetes cluster with Dask workers so that any operation on a NumPy array is transparently and automatically executed over multiple nodes and their CPUs.

Advantages of using Dask:

Designed to parallelize the Python ecosystem

- Flexible parallel computing paradigm;
- Parallelize existing Python code in transparent way for the users (no need to adapt the code)
- Co-developed with Pandas/SKLearn/Jupyter teams
- Ideal for analysis of large raster files

Scales

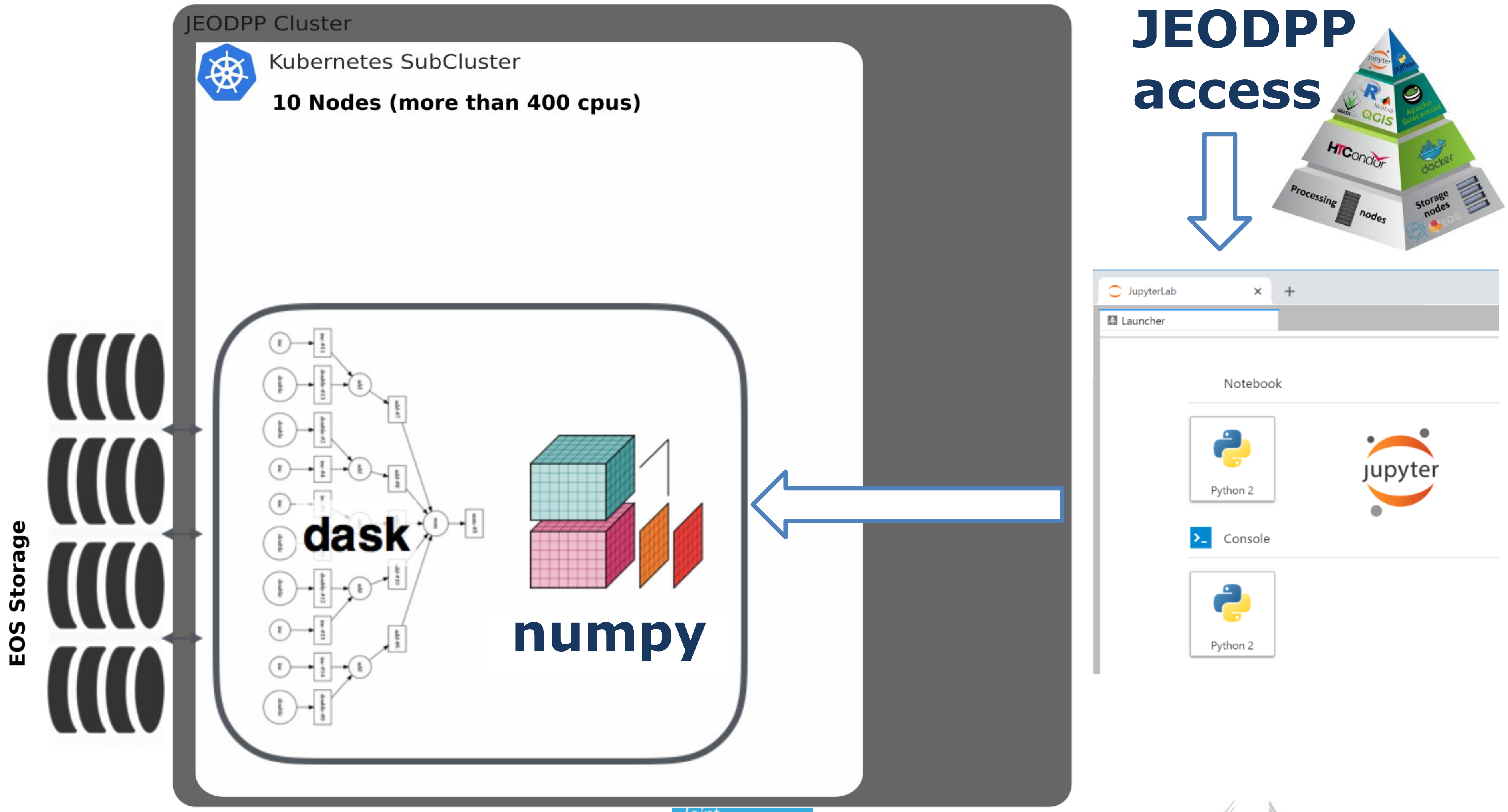
- Scales from multicore to 1,000-node clusters
- Resilience, responsive, and real-time

Use Cases

- DEM analysis for Erosion Index over all Europe

- Metereological raster analysis for forecasting purposes

Dask on Kubernetes/JEODPP



JEODPP Interactive visualisation and analysis: JEO-lab

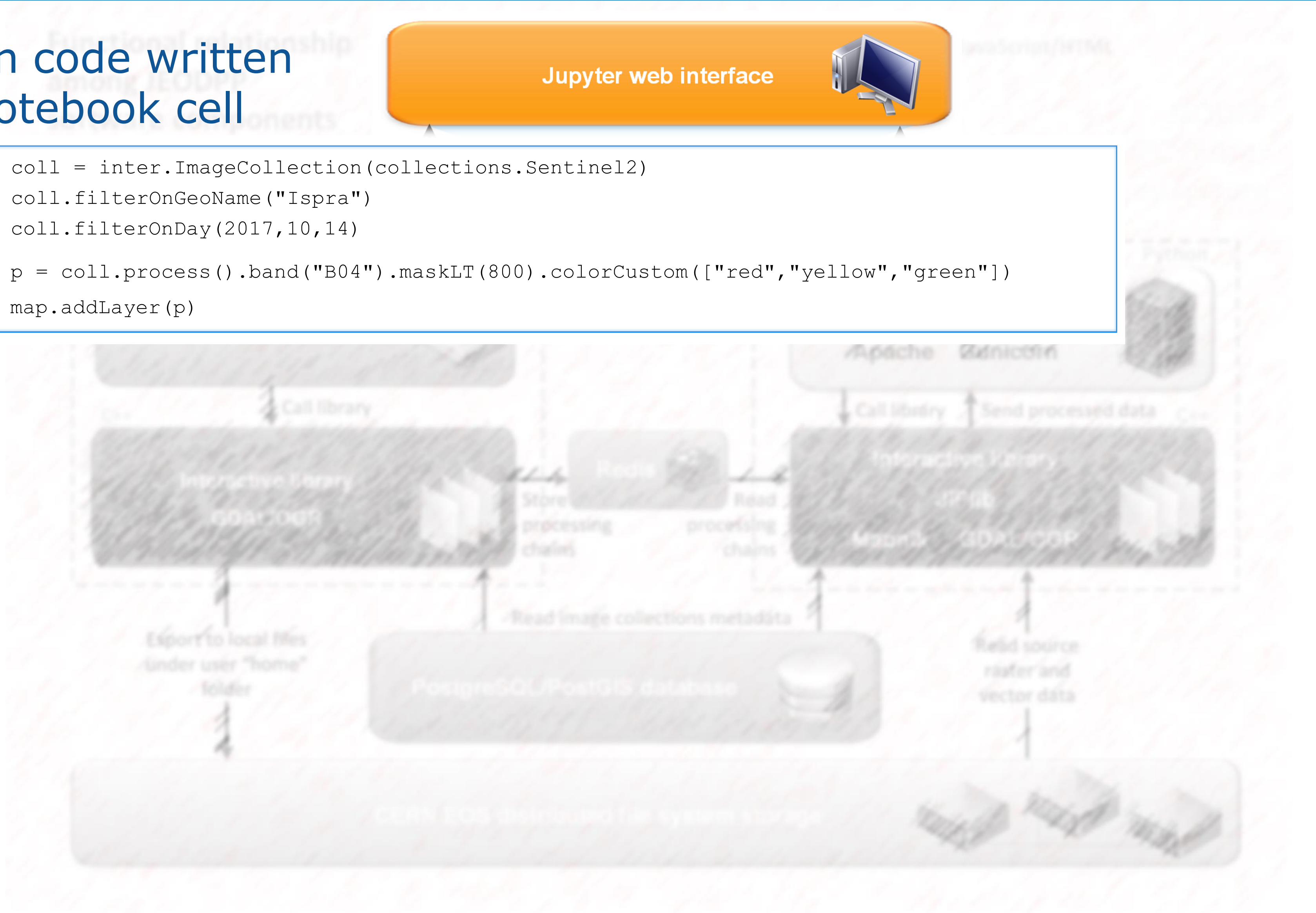
- Web interface to visualize and analyze big geospatial data
- Allows fast search and display of complex dataset
- Creates an agile test environment for raster and vector processing algorithms thanks to the immediate display of the output results
- Available for geospatial expert with some programming capabilities
- Allows easy creation of GUI applications for non programmers (ipywidgets, ipyleaflet, bqplot, qgrid)
- Extension in near future to app-mode using Voila

Deferred processing

- No data is pre-calculated, similarly to Google Earth Engine platform
- **Processing steps,** their input parameters and their combinations in **processing chains** are defined by the user in the python client environment and saved in JSON format
- Processing chains are executed server side in a highly parallel infrastructure by an image processing C++ library having a direct access to data: **TileEngine**
- Results are sent via HTTPS to the **ipyleaflet** client

Deferred processing in action

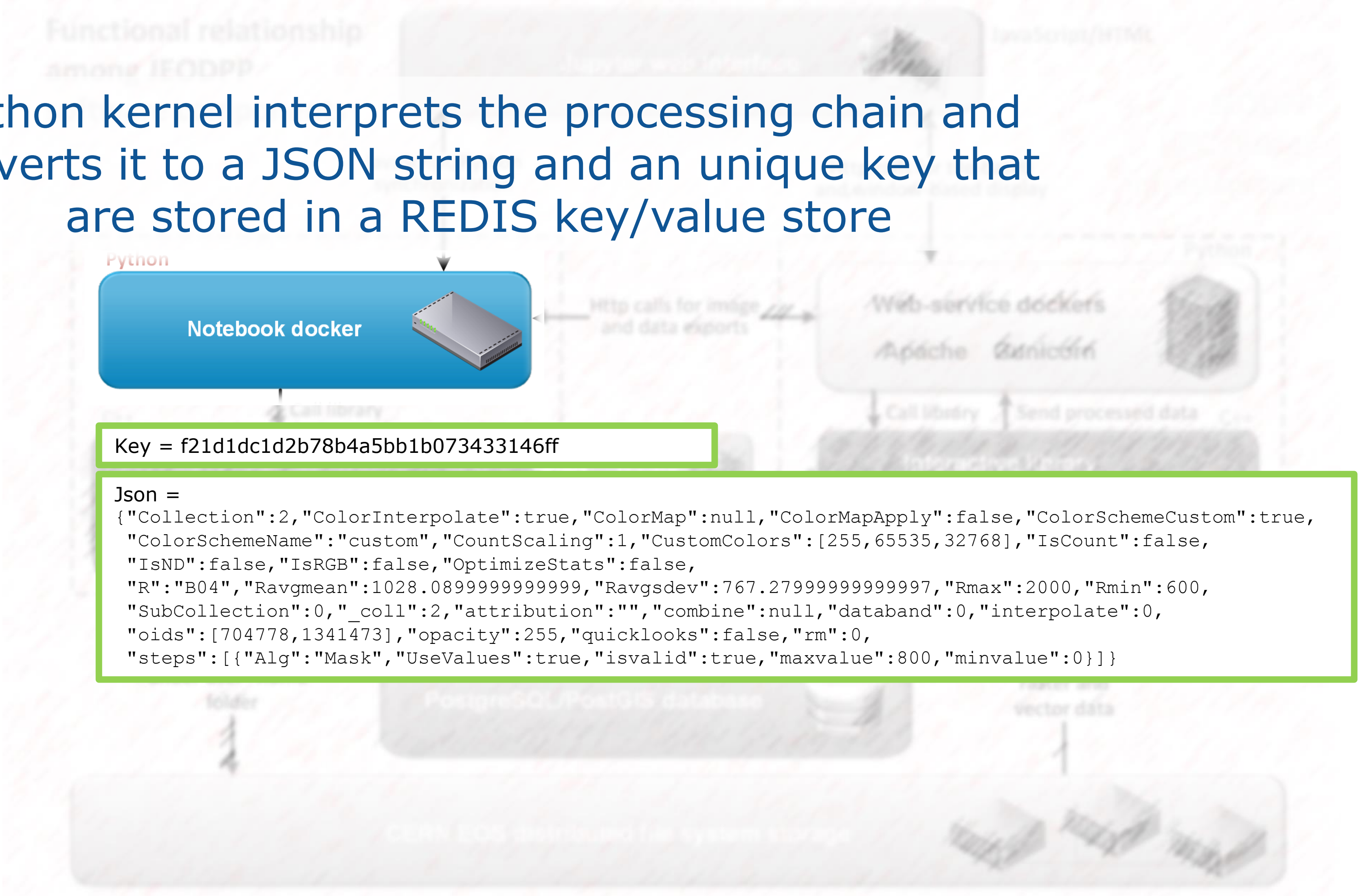
Python code written in a notebook cell



```
In [ ]:  coll = inter.ImageCollection(collections.Sentinel2)
         coll.filterOnGeoName("Ispra")
         coll.filterOnDay(2017,10,14)

         p = coll.process().band("B04").maskLT(800).colorCustom(["red","yellow","green"])

         map.addLayer(p)
```

Deferred processing in action

Python kernel interprets the processing chain and converts it to a JSON string and an unique key that are stored in a REDIS key/value store

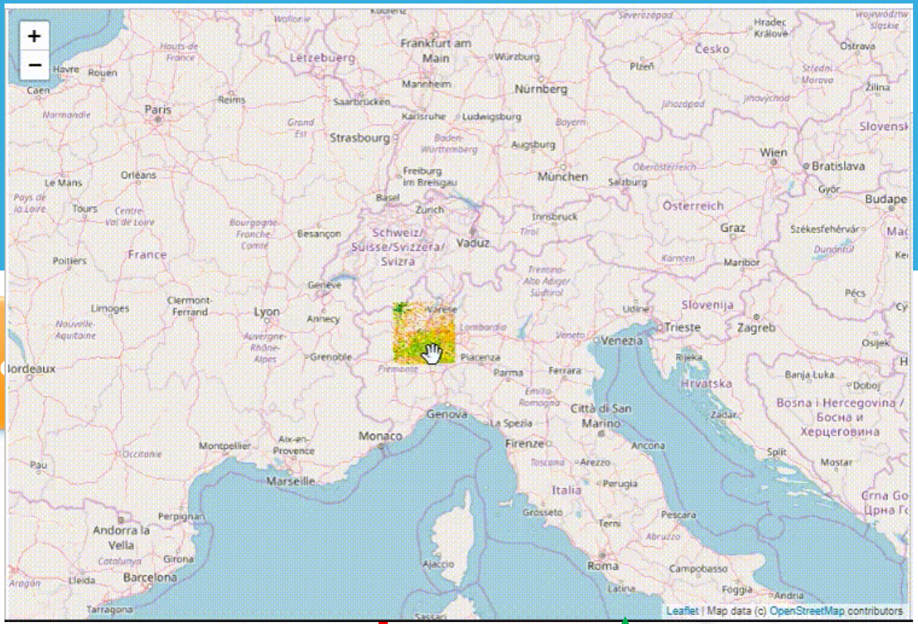


Key = f21d1dc1d2b78b4a5bb1b073433146ff

Json =

```
{"Collection":2,"ColorInterpolate":true,"ColorMap":null,"ColorMapApply":false,"ColorSchemeCustom":true,
 "ColorSchemeName":"custom","CountScaling":1,"CustomColors":[255,65535,32768],"IsCount":false,
 "IsND":false,"IsRGB":false,"OptimizeStats":false,
 "R":"B04","Ravgmean":1028.0899999999999,"Ravgsdev":767.27999999999997,"Rmax":2000,"Rmin":600,
 "SubCollection":0,"_coll":2,"attribution":"","combine":null,"databand":0,"interpolate":0,
 "oids":[704778,1341473],"opacity":255,"quicklooks":false,"rm":0,
 "steps":[{"Alg":"Mask","UseValues":true,"isvalid":true,"maxvalue":800,"minvalue":0}]}
```

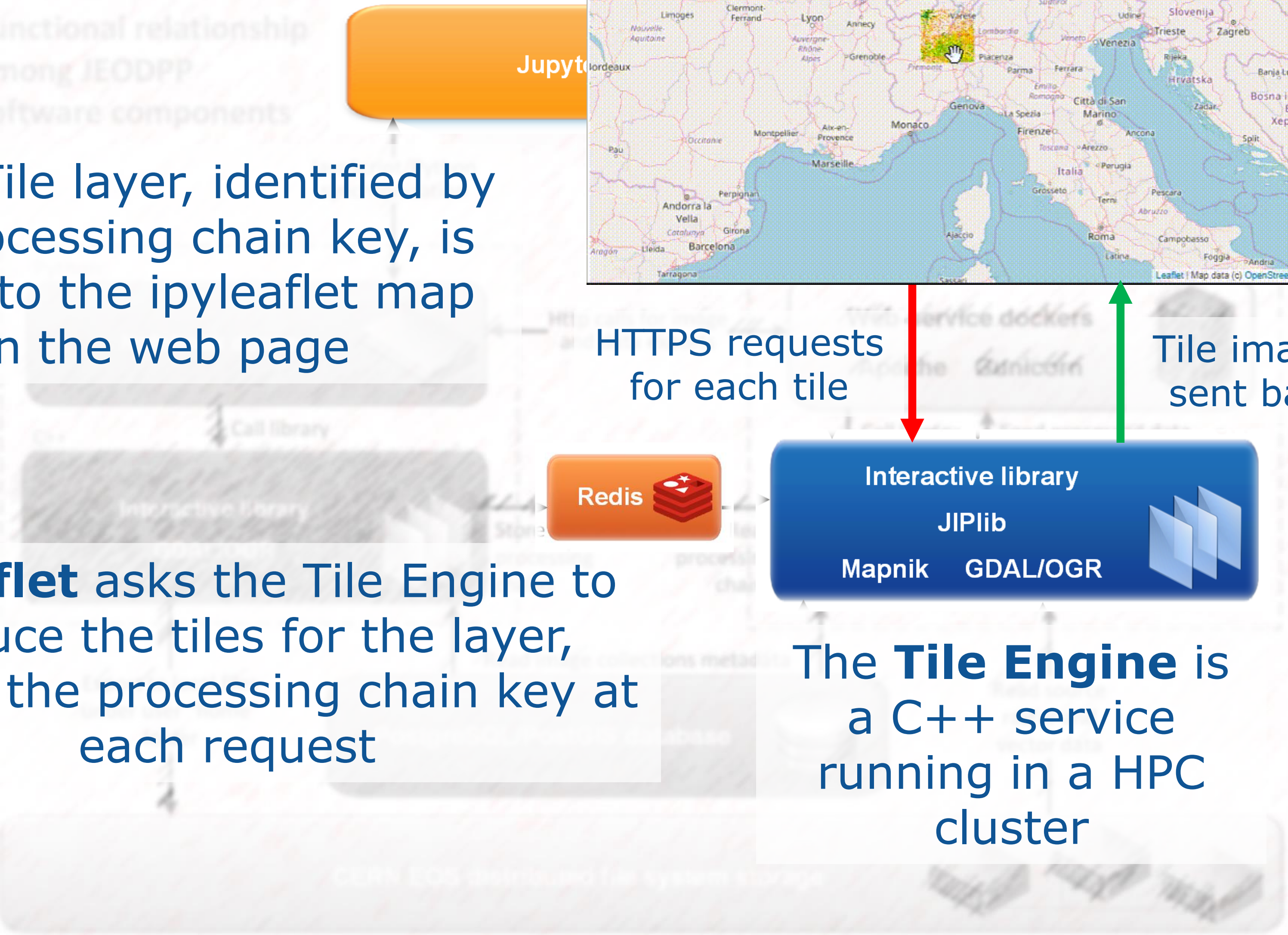
Deferred processing



A new Tile layer, identified by the processing chain key, is added to the ipyleaflet map on the web page

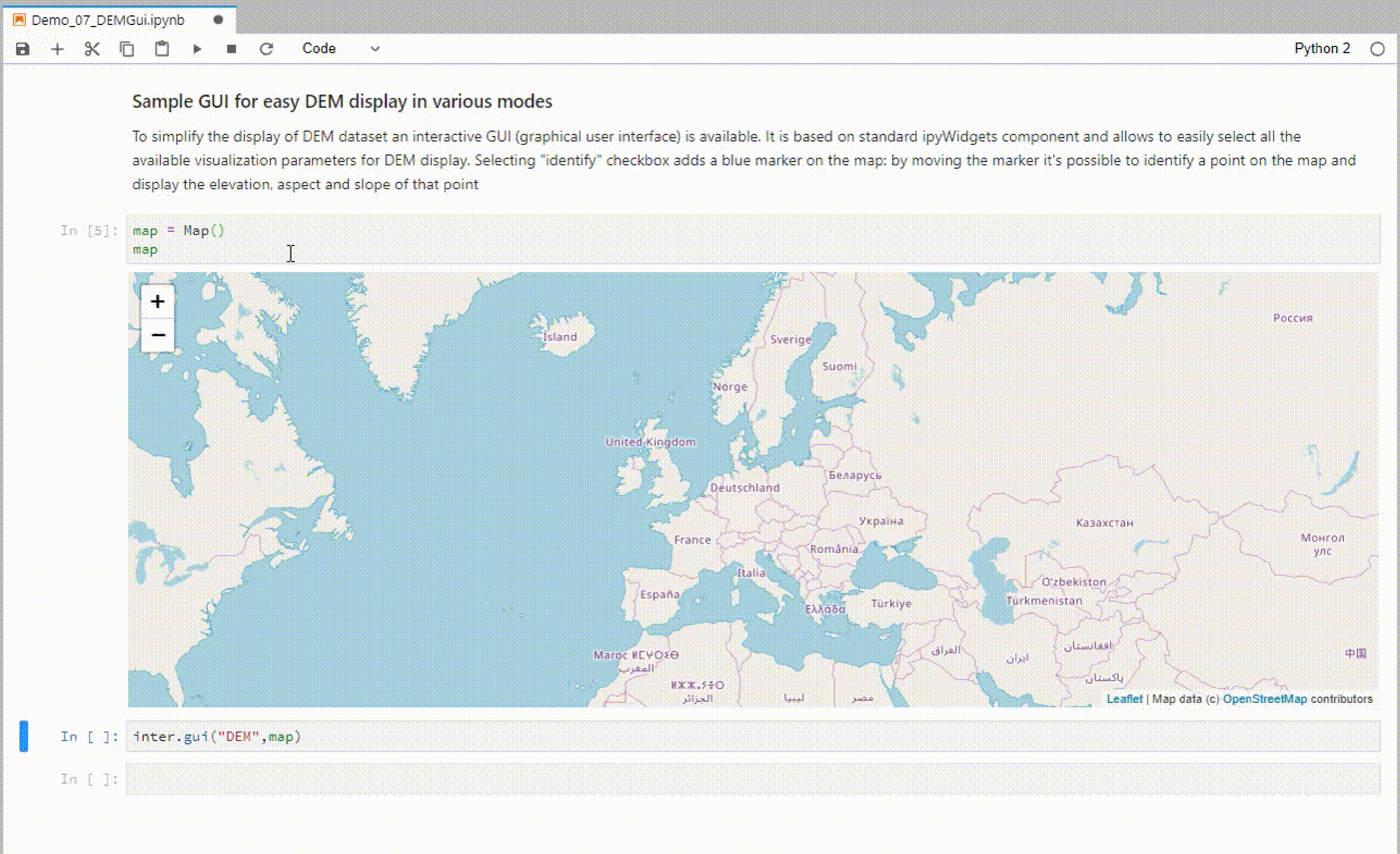
HTTPS requests for each tile

Tile images sent back



ipyleaflet asks the Tile Engine to produce the tiles for the layer, passing the processing chain key at each request

The **Tile Engine** is a C++ service running in a HPC cluster



Demo_07_DEMGui.ipynb

Code

Python 2

Sample GUI for easy DEM display in various modes

To simplify the display of DEM dataset an interactive GUI (graphical user interface) is available. It is based on standard ipyWidgets component and allows to easily select all the available visualization parameters for DEM display. Selecting "identify" checkbox adds a blue marker on the map: by moving the marker it's possible to identify a point on the map and display the elevation, aspect and slope of that point

```
In [5]: map = Map()
        map
```

```
In [ ]: inter.gui("DEM",map)
```

```
In [ ]:
```

JEODPP
The JRC Big Data Platform

One step further: Python code injected server-side

Need to increase user flexibility

Need to use available python libraries

Why not allow to inject custom python code to the server-side processing chain Tile Engine running in the HPC?

Function definition and **function call** are converted to strings using python **inspect** module (getsource and getcallargs functions)

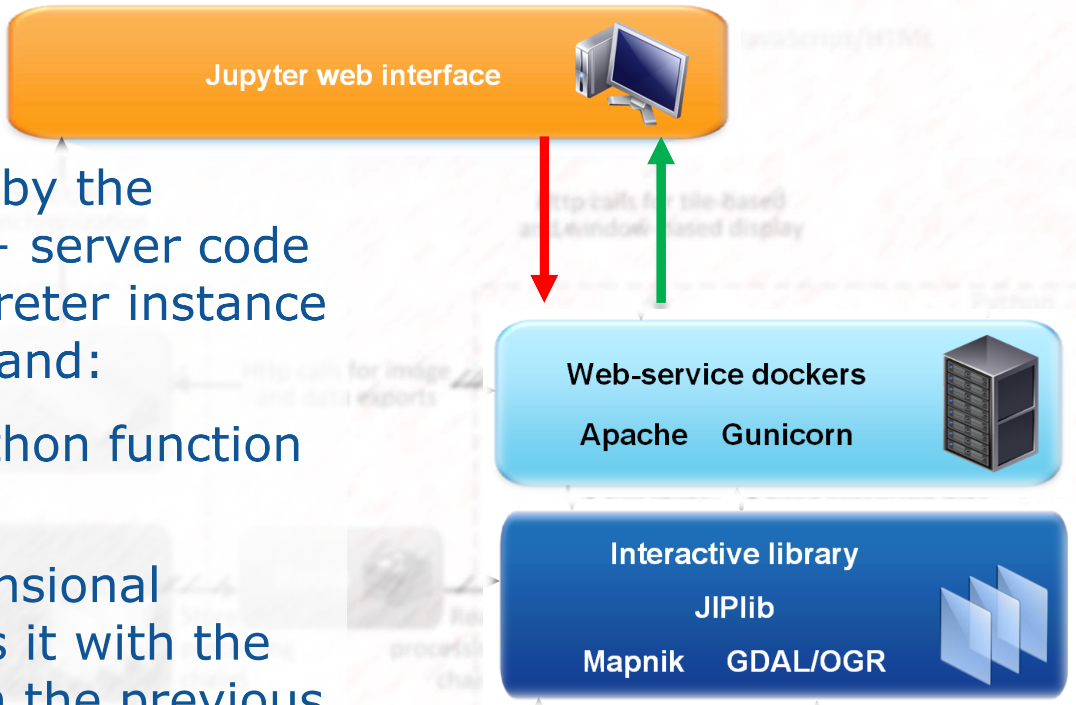
A special processing step, called **execute**, was implemented

For security reasons the list of available libraries is limited but customizable on-demand

Joint Research Centre

European Commission

Python code injected server-side



For each tile requested by the ipyleaflet map, the C++ server code creates a python interpreter instance (**python embedding**) and:

- executes in it the python function definition
- creates a multi-dimensional Numpy array and fills it with the results obtained from the previous steps of the precession chain
- executes the python function call
- Reads the result returned by the user function and passes it to the next step of the chain

```
def maskpy(img, n):
    return img[img<=n] = 0
```

An example: stubble burning mapping

Courtesy: JRC Directorate D Sustainable Resources, **D.5** Food Security

- Deliberate setting fire of the straw stubble that remains after wheat and other grains have been harvested.
- The practice was widespread until the 1990s, when governments increasingly restricted its use
- Many risks:
  - *Pollution from smoke*
  - *Risk of fires spreading out of control*
  - *…*

Detection of stubble burning from satellite images using python code injected server-side

Stubble burning detection for Sentinel2:

B04 < 1000 AND
B06 < 1200 AND
B08 < 1200 AND
B11 > 500 AND
B11 < 1600

Function definiton that implements the stubble burning algorithm:

```
def stubble(img, v4, v6, v8, v11min, v11max):
    b4,b6,b8,b11 = img[0],img[1],img[2],img[3]
    res = numpy.ones_like(b4)
    res[b4>=v4] = 0
    res[b6>=v6] = 0
    res[b8>=v8] = 0
    res[numpy.logical_or(b11<=v11min,
                         b11>=v11max)] = 0
    return res
```

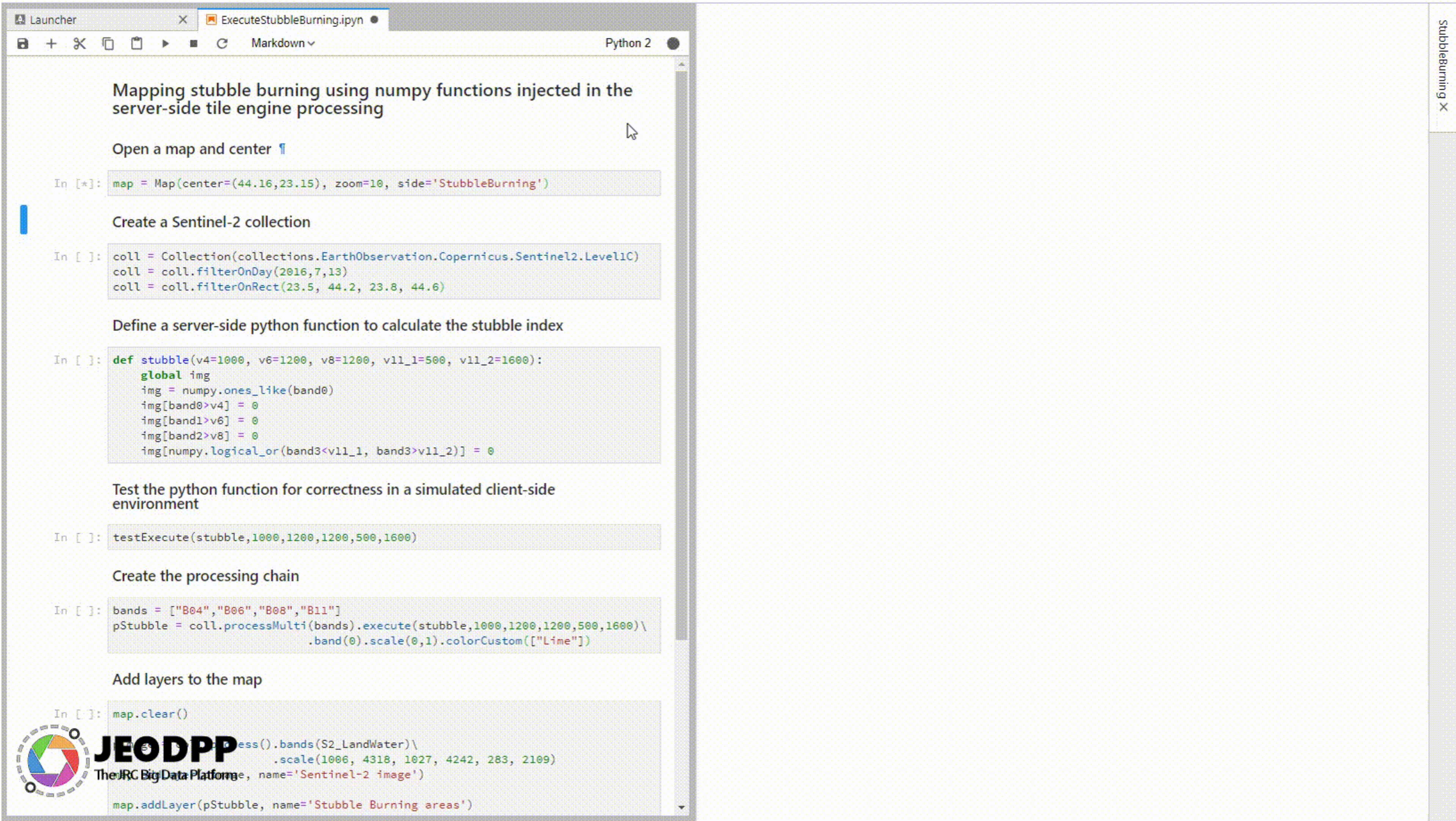
Multi-band processing chain containing the python function call:

```
p = coll.processMulti(["B04","B06","B08","B11"])
        .execute(stubble,1000,1200,1200,500,1600)
        .band(0).scale(0,1).colorCustom(["Lime"])
```

Joint Research Centre

European Commission

Detection of stubble burning from satellite images using python code injected server-side



Mapping stubble burning using numpy functions injected in the server-side tile engine processing

Open a map and center ¶

```
In [*]: map = Map(center=(44.16,23.15), zoom=10, side='StubbleBurning')
```

Create a Sentinel-2 collection

```
In [ ]: coll = Collection(collections.EarthObservation.Copernicus.Sentinel2.Level1C)
        coll = coll.filterOnDay(2016,7,13)
        coll = coll.filterOnRect(23.5, 44.2, 23.8, 44.6)
```

Define a server-side python function to calculate the stubble index

```
In [ ]: def stubble(v4=1000, v6=1200, v8=1200, v11_1=500, v11_2=1600):
            global img
            img = numpy.ones_like(band0)
            img[band0>v4] = 0
            img[band1>v6] = 0
            img[band2>v8] = 0
            img[numpy.logical_or(band3<v11_1, band3>v11_2)] = 0
```

Test the python function for correctness in a simulated client-side environment

```
In [ ]: testExecute(stubble,1000,1200,1200,500,1600)
```

Create the processing chain

```
In [ ]: bands = ["B04","B06","B08","B11"]
        pStubble = coll.processMulti(bands).execute(stubble,1000,1200,1200,500,1600)\
                                    .band(0).scale(0,1).colorCustom(["Lime"])
```

Add layers to the map

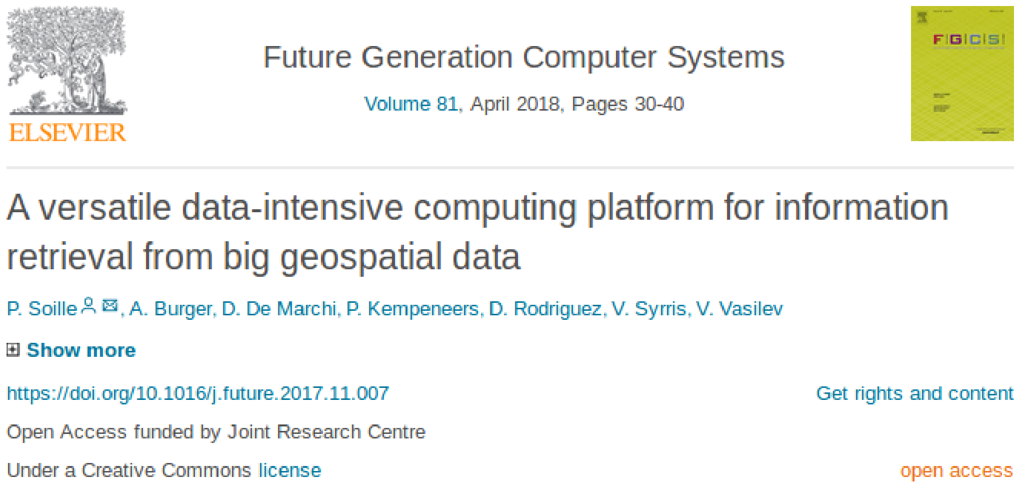
```
In [ ]: map.clear()

        [illegible]ess().bands(S2_LandWater)\
                         .scale(1006, 4318, 1027, 4242, 283, 2109)
        [illegible]e, name='Sentinel-2 image')

        map.addLayer(pStubble, name='Stubble Burning areas')
```

JEODPP
The JRC Big Data Platform

Takeaway message

- Multi-petabyte scale platform
- Versatile environment for Big data visualization, analysis and processing
- With the server-side injection of python code, deferred processing for interactive visualization is even more flexible and open
- Batch and interactive processing are available and will be even more linked in the near future (JEO-lab as a fast prototyping environment to test and create complex batch processing jobs)

Thank you for your attention!



Future Generation Computer Systems

Volume 81, April 2018, Pages 30-40

A versatile data-intensive computing platform for information retrieval from big geospatial data

P. Soille, A. Burger, D. De Marchi, P. Kempeneers, D. Rodriguez, V. Syrris, V. Vasilev

⊞ Show more

https://doi.org/10.1016/j.future.2017.11.007 Get rights and content

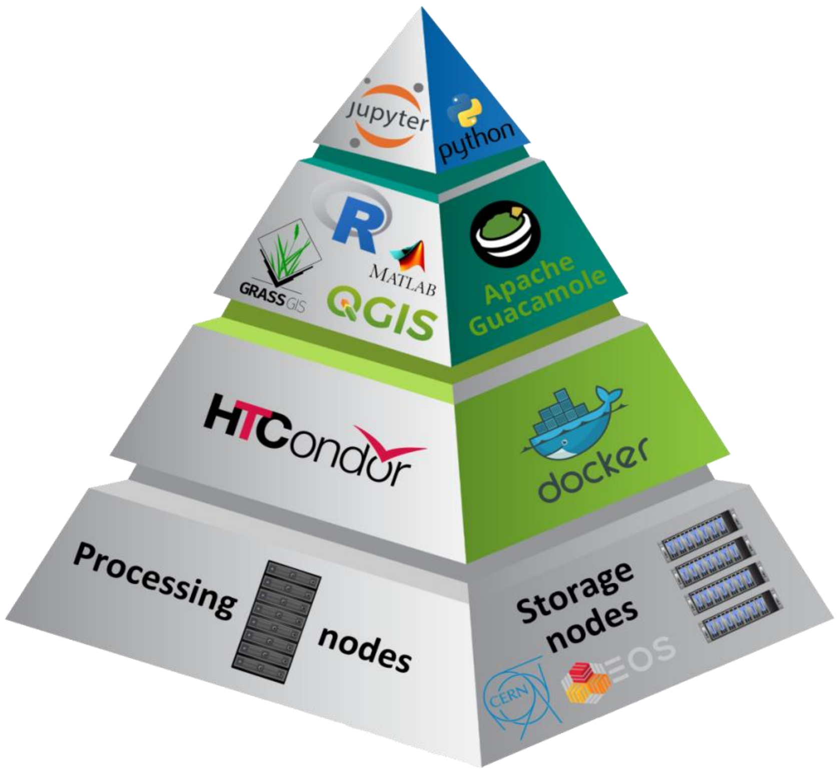
Open Access funded by Joint Research Centre

Under a Creative Commons license open access

https://doi.org/10.1016/j.future.2017.11.007

Publication list:
https://cidportal.jrc.ec.europa.eu/home/publications

JEODPP
The JRC Big Data Platform



EO&SS@BigData pilot project

Unit I.3 Text and Data Mining Unit
Directorate I Competences

Joint Research Centre

European Commission