# AVISO data access and ramp up experiments

G.Dibarboure, N.Picot (CNES)
F.Briol (CLS), L.Gaultier (ODL)

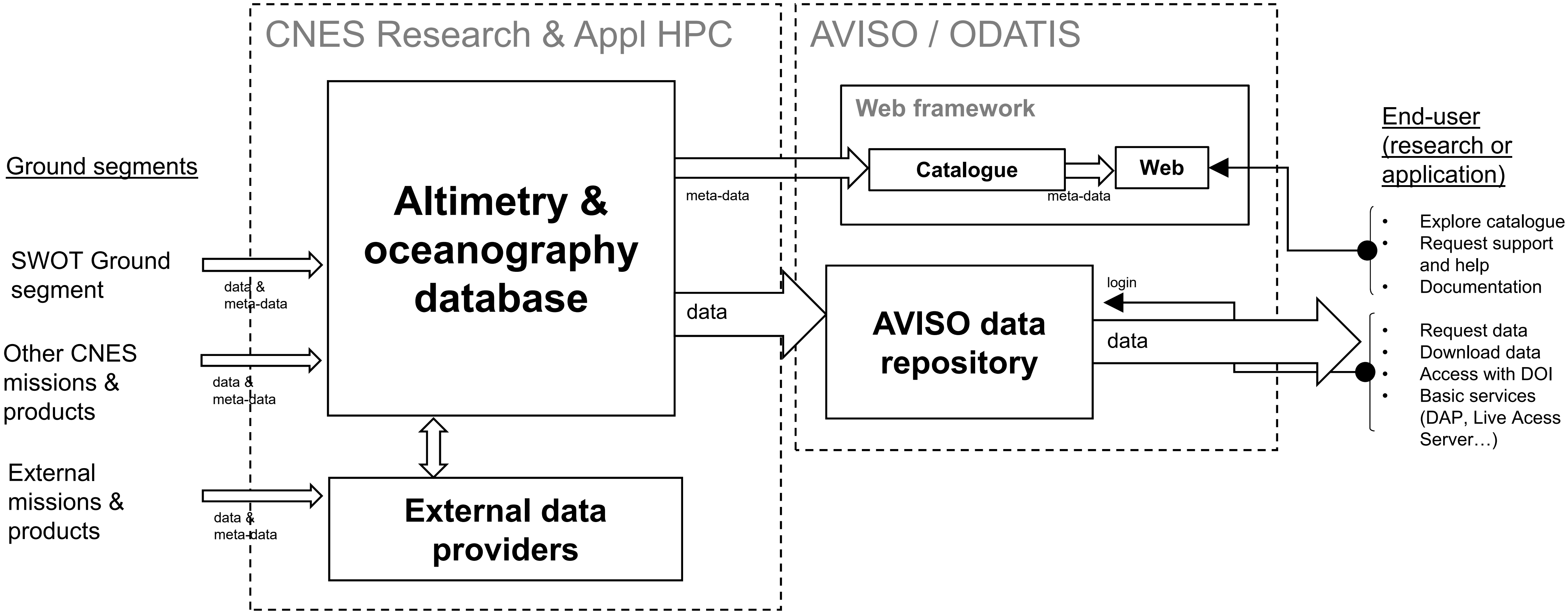
SWOT Science Team Meeting
June 2019

## Introduction

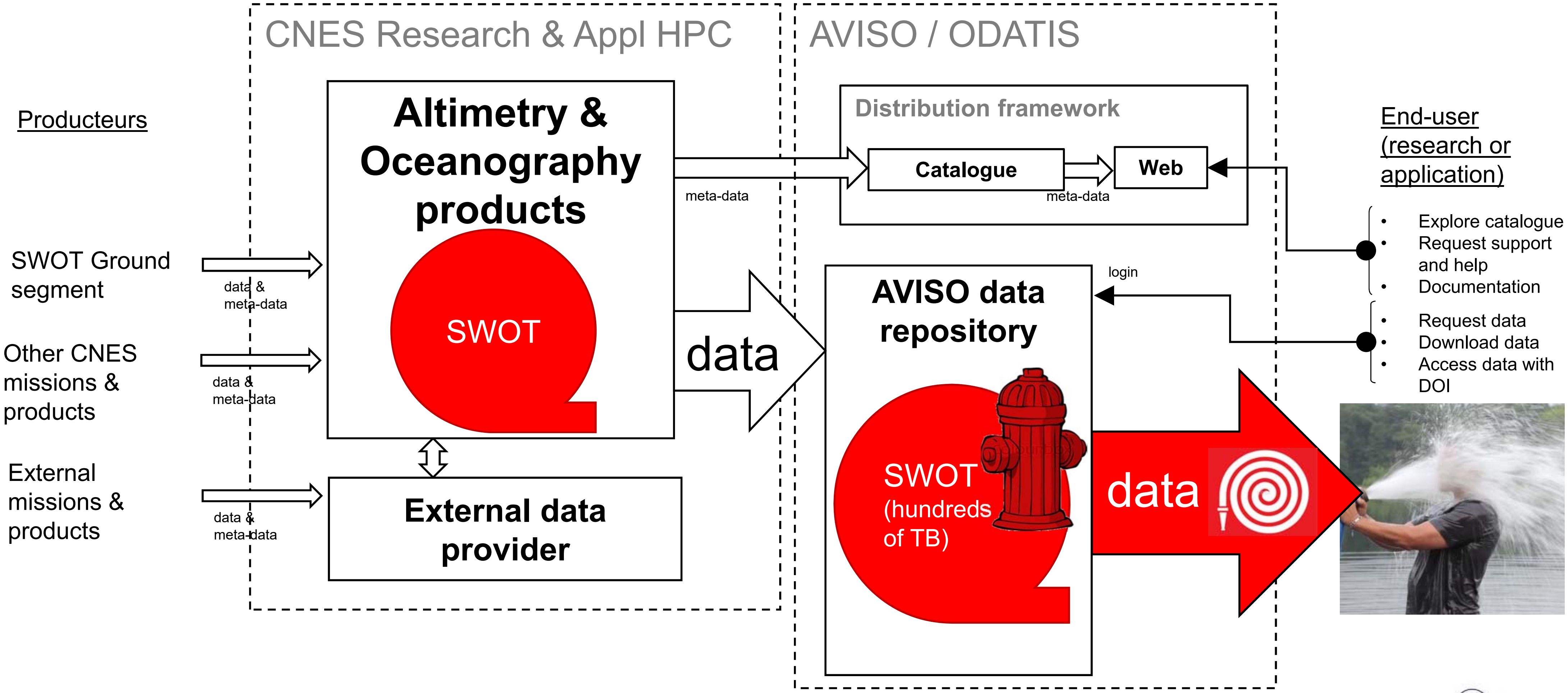
- Science and applications share a common challenge: the learning curve of SWOT
- Basic challenges
  - Data access & download
  - Data volume & processing time
  - SWOT is better when combined with other data
- Practical examples
  - How can I get KaRIN data that are consistent/calibrated with Jason-CS or S3?
  - Can I get match-ups with <other mission> without downloading everything?
  - How do I run <big computation> on 2+ years of SWOT with my modest laptop/server?


- Objective of this talk: show how we are using ocean simulation tool as training wheels

# The old paradigm



# The challenge with SWOT



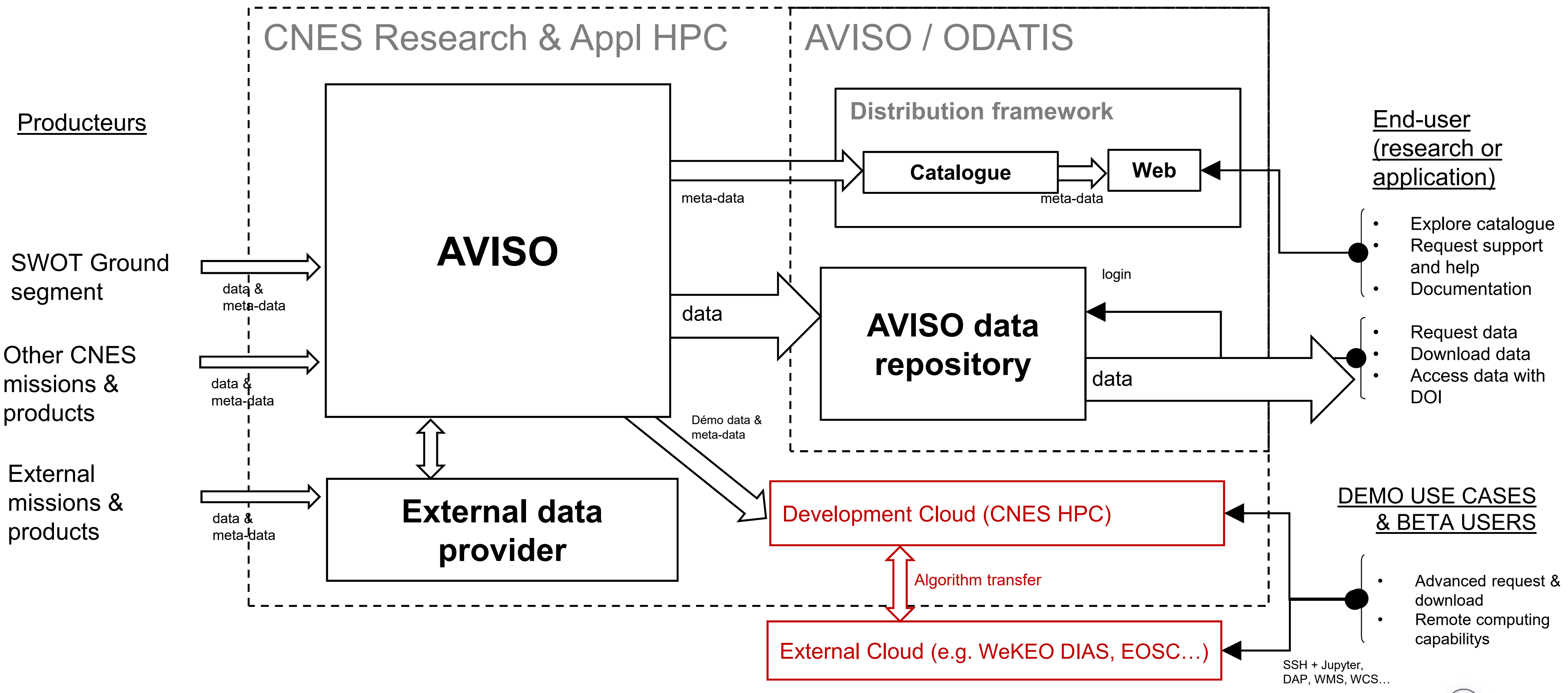
# The new paradigm

- Bring the algorithm to the data, not the opposite

- New technology is <u>the</u> main challenge for the end-user
  - Storage format might seem trivial but it is a critical item
  - Running multi-core computation is also essential and not always simple
  - Exploring data remotely (e.g. simple visualization) requires specific sets of skills and tools

- Some technologies analyzed by CNES
  - PANGEO software stack: promoted by oceanography community (e.g. SWOT science team)
  - DATACUBE used by some so-called DIAS datacenters (ESA Copernicus framework)

# The new paradigm

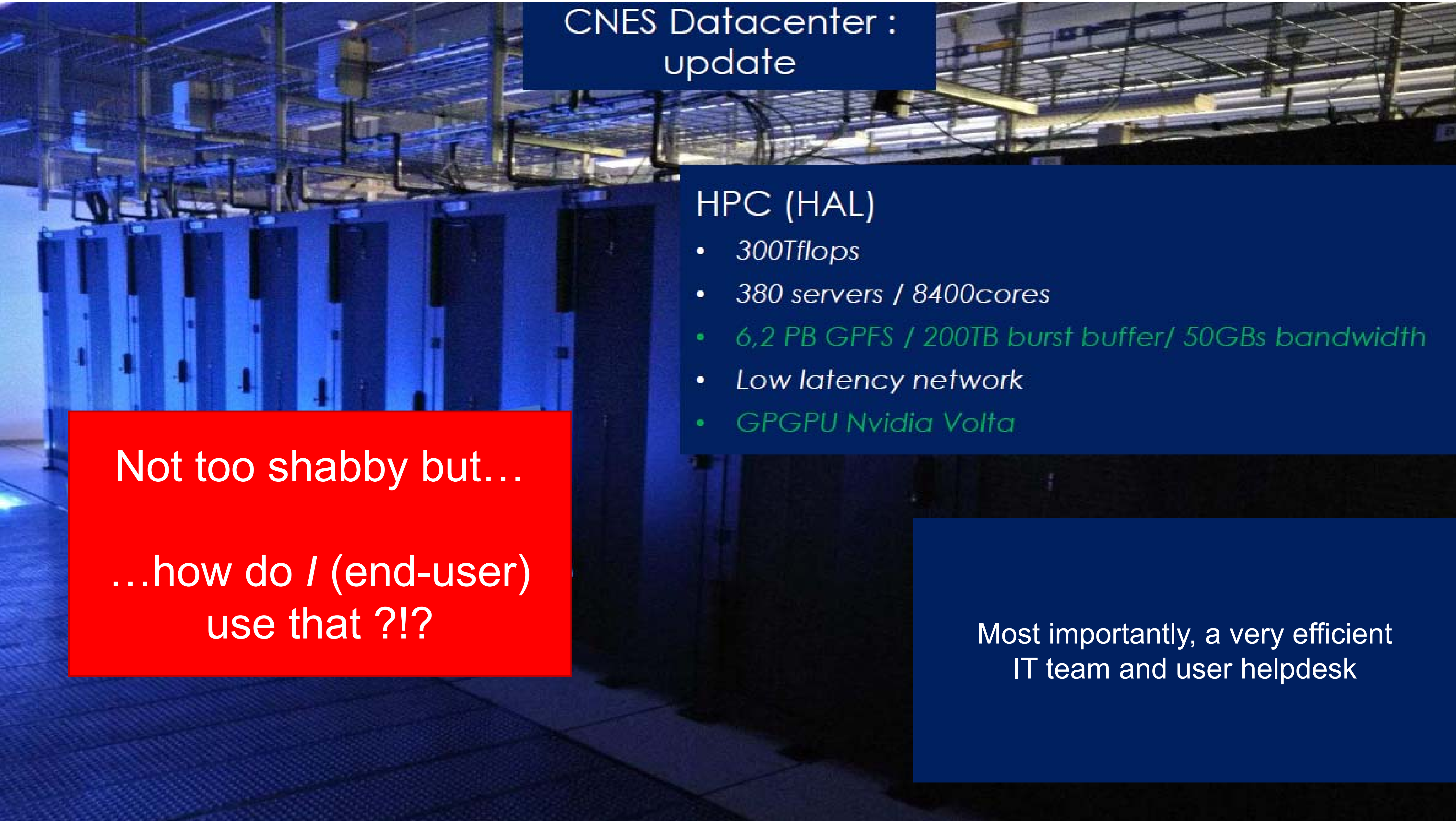


# SWOT proof of concept

- Use-cases
  - Run **temporal** algorithms on HR ocean model stored as **snapshots** (e.g. MITgcm, eNATL60, Mercator)
  - Speed up SWOT's « ocean science simulator » to generate 1 year of simulated data instantaneously
  - Speed up a **multi-temporal** algorithm (e.g. cross-over match-ups) over this large dataset
  - Compute **multi-sensor** match-ups rapidly (e.g. SWOTsim with Sentinel-3 topo and ocean color)
  - Visualize this large dataset **efficiently**

- The experiment's goal is to serve as training wheels
  - **Test technologies** (realistic benchmarks) that are mandatory to speedup algorithms and data access
  - Identifies technical roadblocks and practical difficulties
  - Investigate if one **single** framework can be used for data distribution and remote computing
  - Ensure the solution is **accessible**: the science team and applications must be trained by the end of the Cal/Val phase

# Example #1: making parallel computing accessible



# CNES Datacenter : update

## HPC (HAL)

- *300Tflops*
- *380 servers / 8400cores*
- *6,2 PB GPFS / 200TB burst buffer/ 50GBs bandwidth*
- *Low latency network*
- *GPGPU Nvidia Volta*

Not too shabby but…

…how do *I* (end-user) use that ?!?

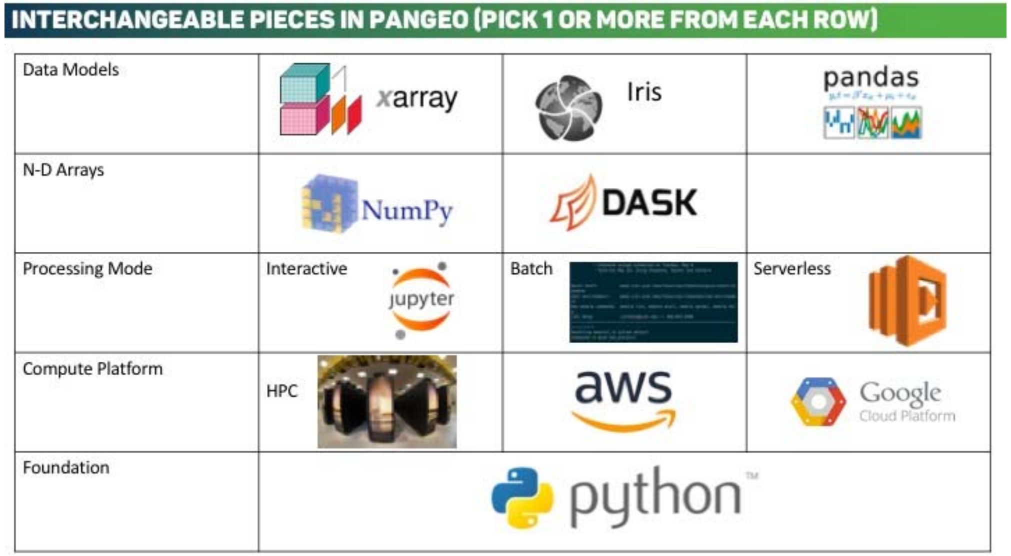
Most importantly, a very efficient IT team and user helpdesk

# Ongoing experiment: PANGEO framework

- Python framework well suited for geoscience science (incl. oceanography)

http://pangeo.io/quickstart.html

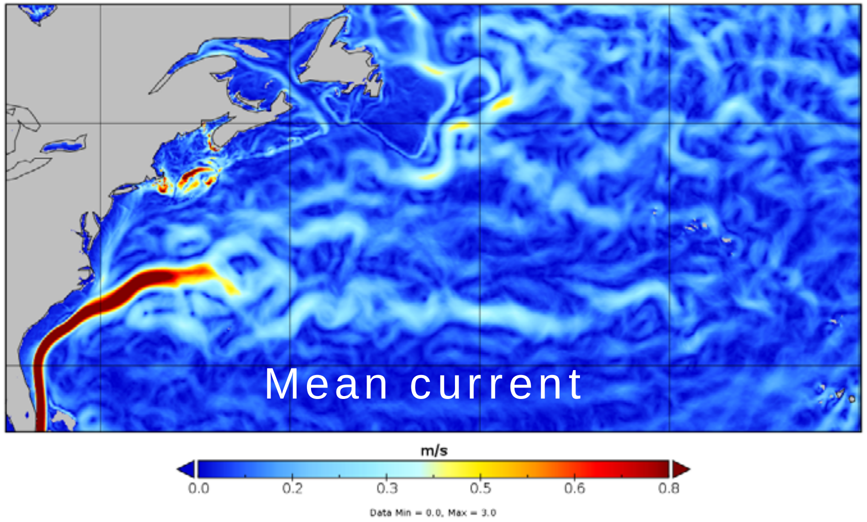
- Suites of libraries for HPC (also work on your personal laptop/server for development)

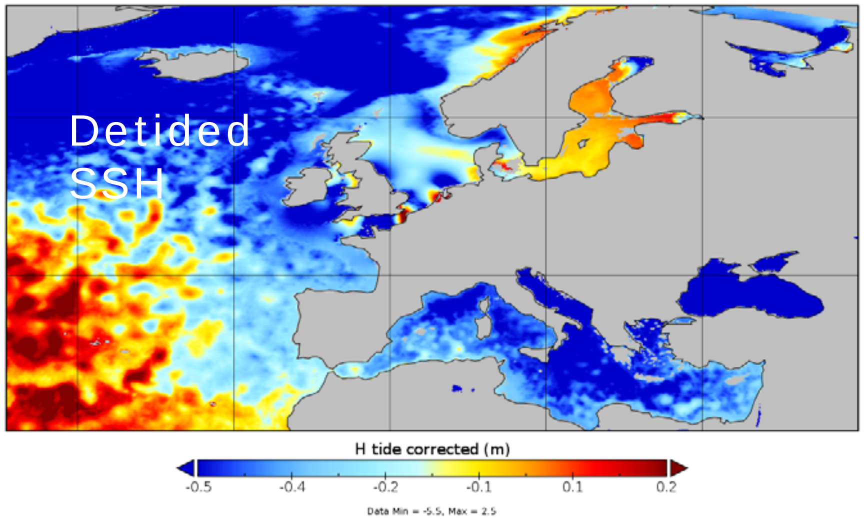


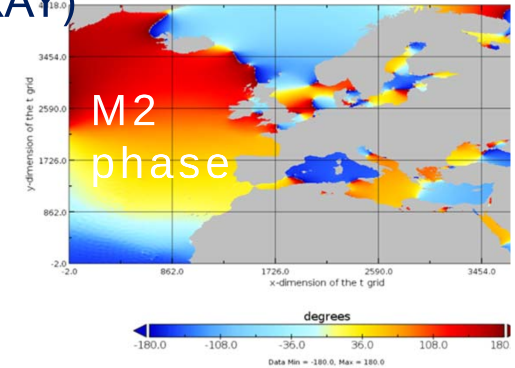

- Good for scientists: SWOT science team members involved in PANGEO community
- Active involvement of CNES IT team members (read: in-house expert support)
- Practical goal: test data storage format & massive parallel computing in SWOT context

# Example: your algorithm should not be throttled by I/O

- Before : process killed after one day
- After :
  - Time average in of U/V in 8 min
  - Tides (all constituents) estimated and removed from MITgcm in a less than 5 hours
- Essential assets
  - Fast I/O (ZARR)
  - Parallelization (DASK)
  - Geoscience interfaces (XARRAY)
- Most importantly, the code **is** accessible to regular end-users (not just developers)



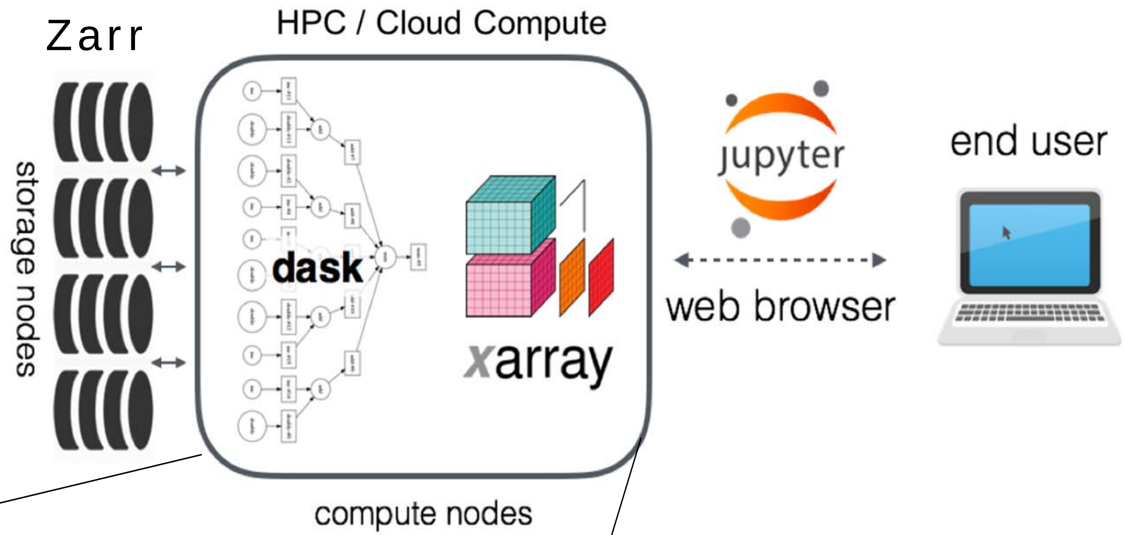



# Use-case: remove tides signal from HF ocean model snapshots

Key numbers
- 30 TB worth of test data
- 1.5 km resolution, global ocean (MITgcm)
- 24 workers
- 3 TB of RAM (all workers)
- 1 single notebook for laptop and HPC with PBS

Notebook execution time
- 4 s to interpolate snapshots on regular grids
- 8 min to perform time average (18 months, hourly)
- 5 hours for a complex algorithm (detiding)




Parallel read of data chunks efficiently (stored as 2D lon/lat snapshots, but 1D time series needed)

Read time series
Harmonic analysis
Write output

Split that computation on N computing nodes

Open web brower…
… use my detiding algorithm on MITgcm H/U/V snapshots

Run algorithm on chunks

Compile outputs from data chunks

min / hours
not days

# Example: interpolating irregular datasets

- Interpolation is trivial when the input dataset is gridded
- It becomes increasingly difficult for irregular grids / pixel clouds… especially when they are dense
- Can we easily interpolate irregular 1.5 km grids?

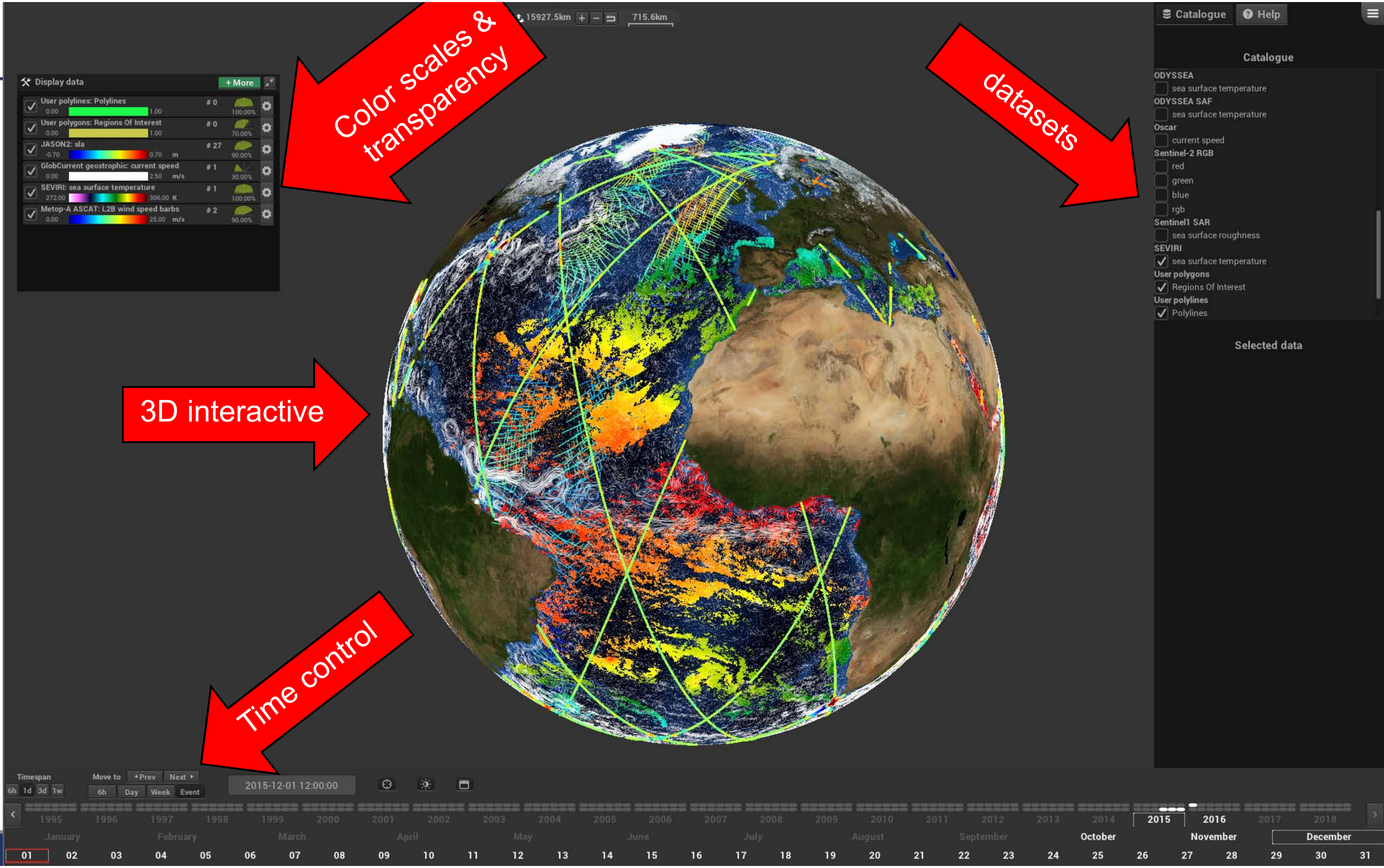
- Before (basic scipy algorithm) : hours to build and apply the KdTree → offline computation
- After: 5 s to re-interpolate a model snapshot on any grid → can be done interactively and on-the-fly
- How : add python API to library developed by CNES/CLS for the SWOT ground segment (GECO)

- Make it simple ! (2 lines to setup the input grid search tree, 1 line to activate the interpolator)
- This « trivial » capability can help end-users do very frequent tasks (e.g. multisensor match-ups)

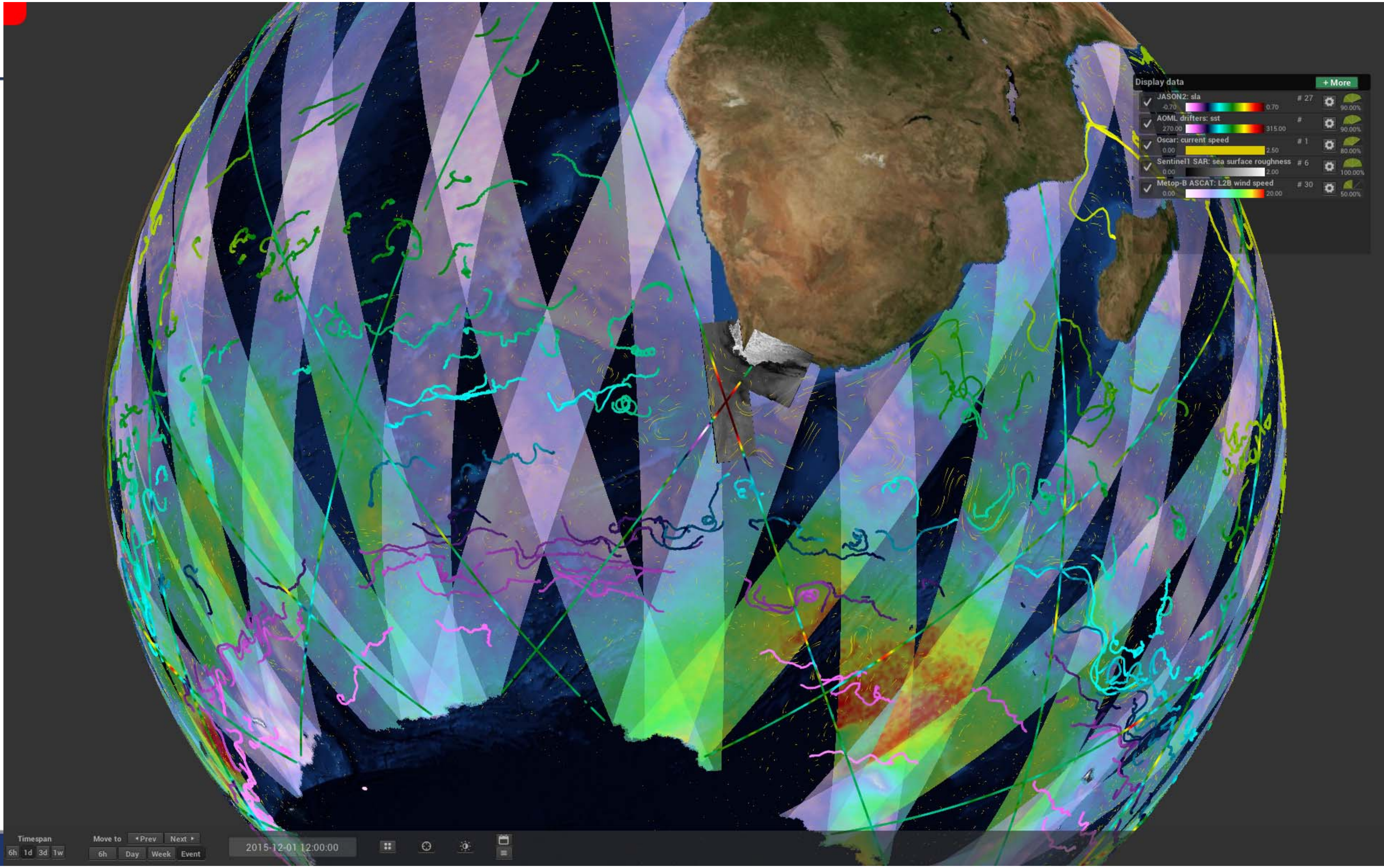
# Example #2: visualization with SEASCOPE

# What is SEASCOPE

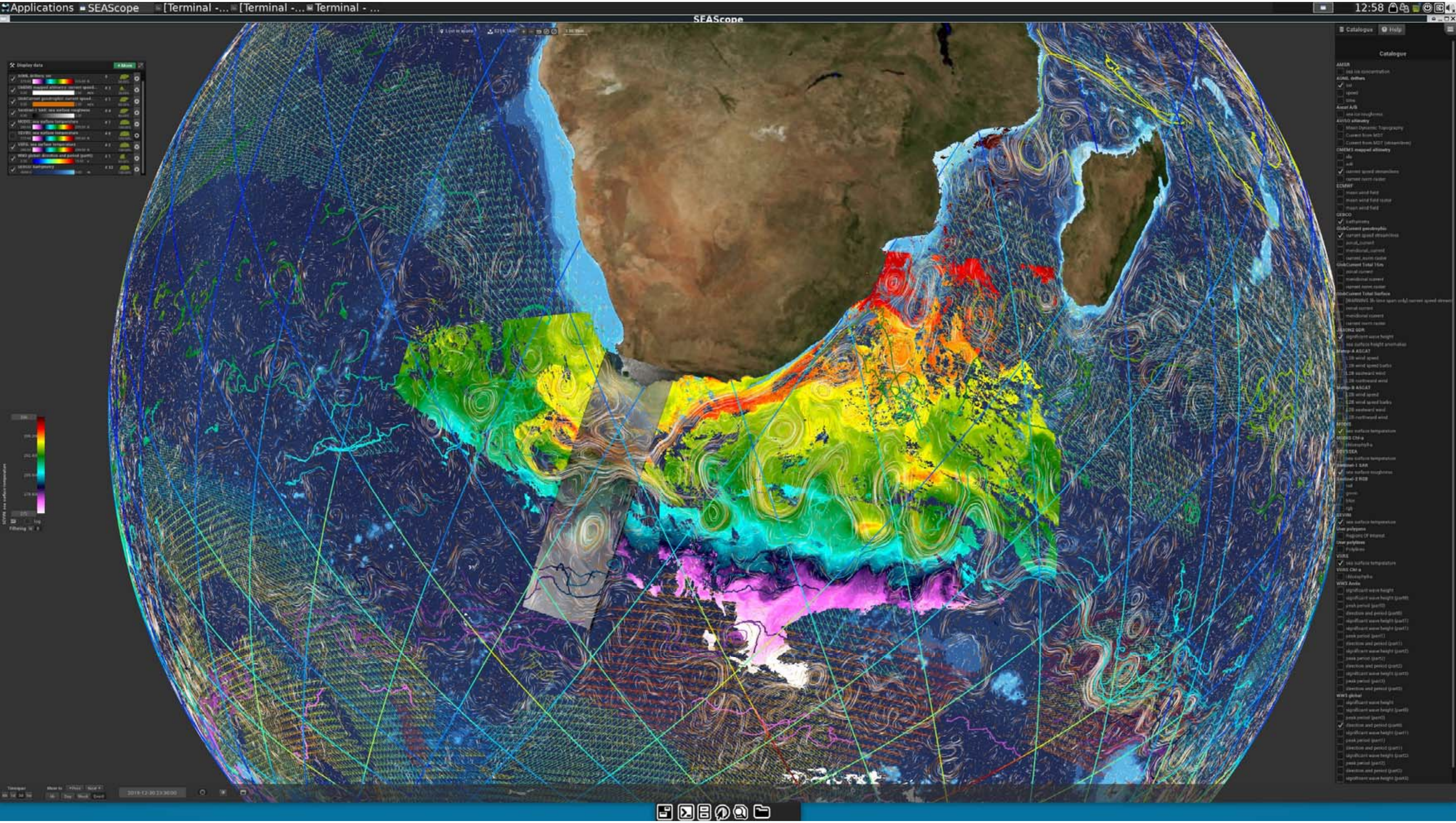
- Data visualization and data exploration software (win, max, linux)
- Developed par Ocean Data Lab (open source, first version already available)
- Beta version already used successfully in ESA training sessions and summer schools
- First release candidate next Fall
- Why SEASCOPE ?
  - Developed by, with, and for the community (not a generic software development)
  - Flexible and reactive (reads native format) with intuitive GoogleEarth feel (attractive and glossy look)
  - Very modular : readers for external data access, in-situ…
  - Can be controlled manually or through python bindings (display tool for Jupyter/PANGEO notebooks)
- Some links
  - Main page and documentation: https://seascope.oceandatalab.com/
  - Jupyter notebook example (camera control) : https://seascope.oceandatalab.com/tutorials/camera.ipynb
  - Video example to explore ocean remote sensing data : https://www.youtube.com/watch?v=zSrfWoxG_FQ

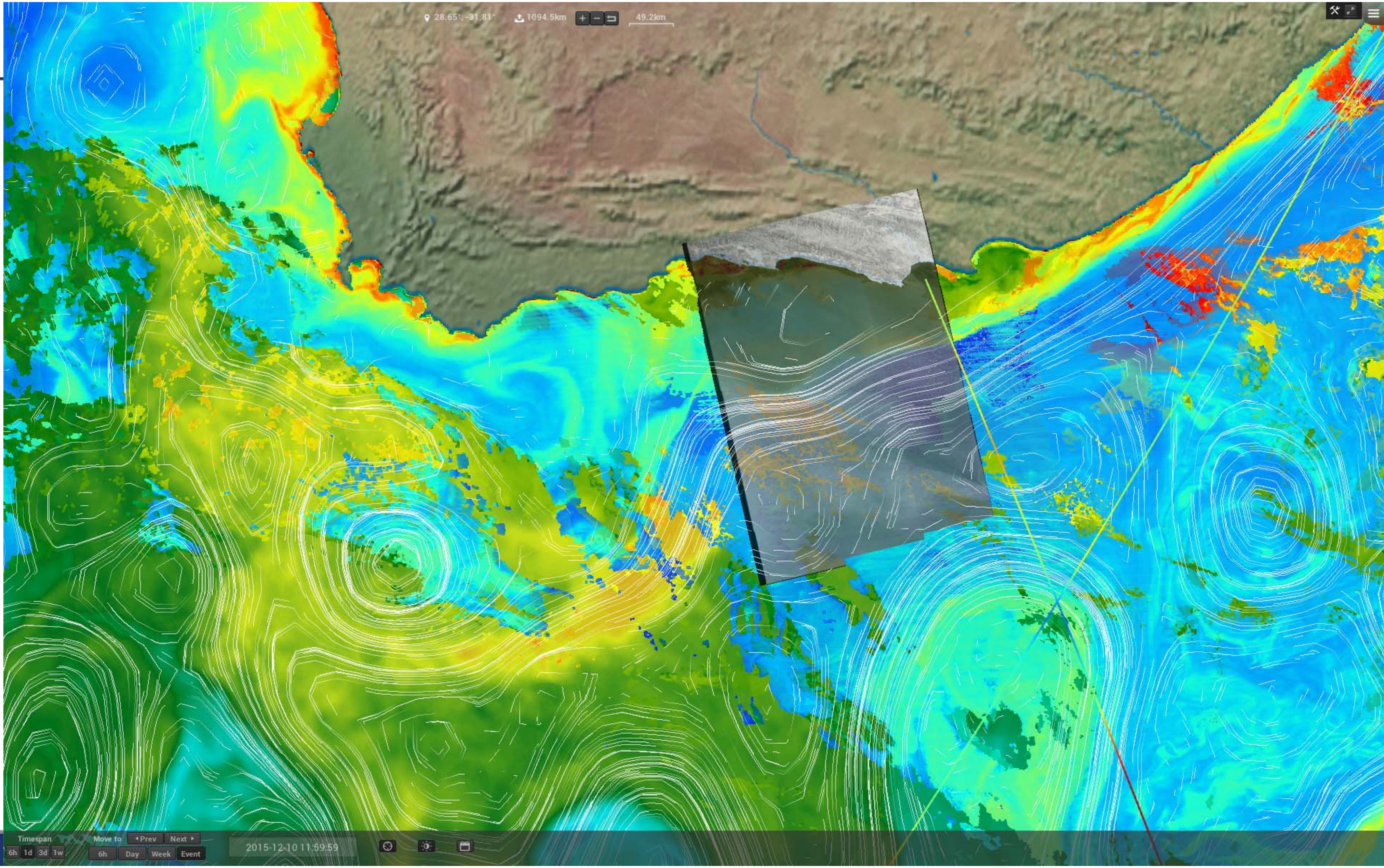


Color scales & transparency
datasets
3D interactive
Time control
Display data
+ More
User polylines: Polylines
User polygons: Regions Of Interest
JASON2: sla
GlobCurrent geostrophic: current speed
SEVIRI: sea surface temperature
Metop-A ASCAT: L2B wind speed barbs
15927.5km
715.6km
Catalogue
Help
Catalogue
ODYSSEA
sea surface temperature
ODYSSEA SAF
sea surface temperature
Oscar
current speed
Sentinel-2 RGB
red
green
blue
rgb
Sentinel1 SAR
sea surface roughness
SEVIRI
sea surface temperature
User polygons
Regions Of Interest
User polylines
Polylines
Selected data
Timespan
Move to
Prev
Next
6h 1d 3d 1w
6h Day Week Event
2015-12-01 12:00:00
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
January February March April May June July August September October November December
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31



Display data
+ More
JASON2: sla
# 27
-0.70
0.70
90.00%
AOML drifters: sst
#
270.00
315.00
90.00%
Oscar: current speed
# 1
0.00
2.50
80.00%
Sentinel1 SAR: sea surface roughness
# 6
0.00
2.00
100.00%
Metop-B ASCAT: L2B wind speed
# 30
0.00
20.00
50.00%
Timespan
6h 1d 3d 1w
Move to
Prev
Next
6h Day Week Event
2015-12-01 12:00:00


Applications
SEAScope
[Terminal - ...
[Terminal - ...
Terminal - ...
12:58
SEAScope
Catalogue
Help
Catalogue
Display data



1094.5km
49.2km
Timespan
6h 1d 3d 1w
Move to
Prev
Next
6h Day Week Event
2015-12-10 11:59:59


nes

# Outlook and conclusions

# Outlook

- Done: basic framework selected and tools assembled
- Now: end-to-end simulation benchmark for SWOT (MITgcm → SWOT sim → XOVER)
- ETA is this summer
- Next steps
  - Demonstrate capability to run multi-sensor match-ups (Sentinel-3 / Jason-CS) on-the-fly
  - Run additional experiments with first « advanced » users (science team & Project cal/val team)
  - Test capability to distribute dataset (OpenDAP, WMS, WCS…) from same repository & HPC

# Development phases

- Step 1
  - Open CNES datacenter to a few beta users (from science team or applications)
  - Buildup datalake with science and application data of interest (e.g. Copernicus Sentinels)
  - Test technologies (e.g. external data download, scientific libraries, visualisation…)
  - Start to empower a small set of advanced users
  - Main goals captured in SWOT proof of concept (see next slides)
- Step 2
  - Ramp up phase with more users and usages
  - Quantify how much support (training, development) and resources (CPU/disk/tools) are needed
  - If needed, identify partner infrastructure (e.g. WeKEO, EOSC) to host SWOT usages and setup M.O.U
  - Consolidate critical elements to encompass a wide range of usages
- Step 3
  - Training of SWOT users
  - Support development of algorithms and research with CNES infrastructure and expert developer helpdesk
  - Common infrastructure for data distribution, advanced services and Cloud computing
  - Transfer larger computational requests (e.g. match-ups with huge Copernicus datasets) to a larger and more operational infrastructure

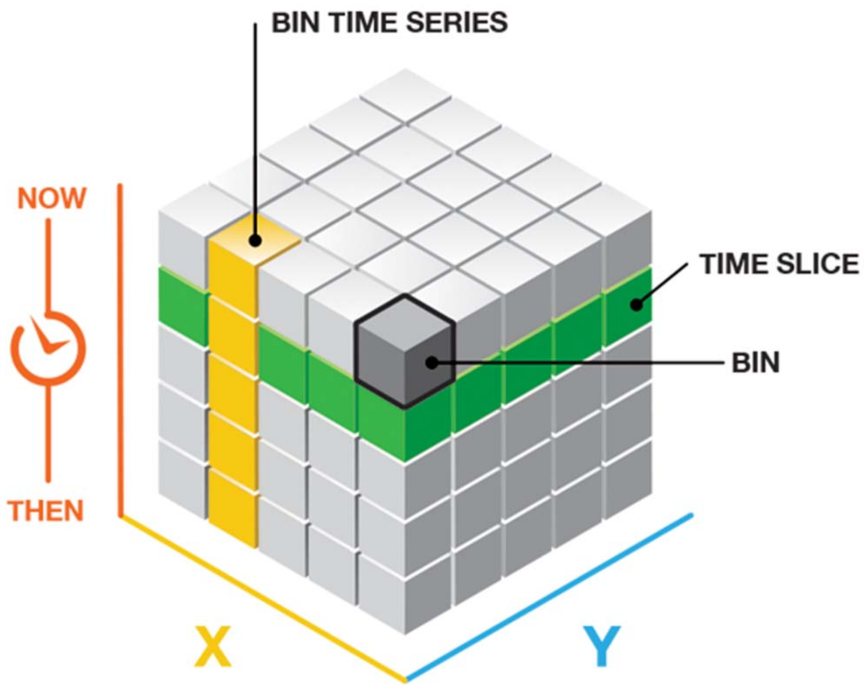
# Conclusions

- Many lessons learned from ongoing experiment with simulated data
  - Trust the framework, empower users, focus on key assets
  - Avoid big software developments (usually not needed and restrictive for researchers & innovation)
- It **is** possible to provide very powerful HPC/HPDA capabilities in a research-friendly framework
  - Reduce download / bandwidth to the minimum
  - Only one copy of the largest datasets
  - Optimization: huge reduction of computing time (clumsy offline queues → convenient interactive research)
  - Same research tools can be developed locally (laptop/server+data samples) and deployed on HPC/cloud
  - Users are not limited by software (they're still free to do their research & application the way they want)
- This requires
  - a lot of preparation (to identify and remove roadblocks)
  - some end-user training (on the methodology)
  - a very solid support (IT system & developers ready for helpdesk) to empower new users or applications
- When learning, the Devil is in the Details (a.k.a helpdesk hotline on speed dial = hours saved)

Thanks for your attention !

BACKUP SLIDES

# Scaling an algorithm with LLC4320 grids




## Naive algorithm

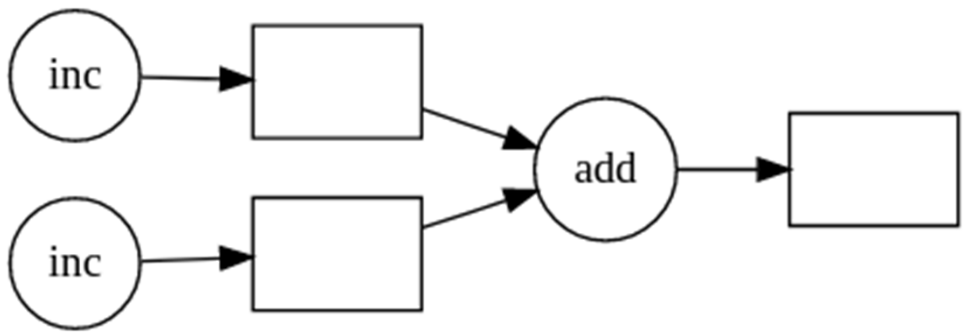
```
for item in time_series(1033):
   for face in range(13):
      for i in range(4320):
         for j in range(4320):
            result[face, i, j] = harmonic_analysis(
               dataset[face, i, j, :],
               f, v0u)
```

Volume to be processed: 2TB

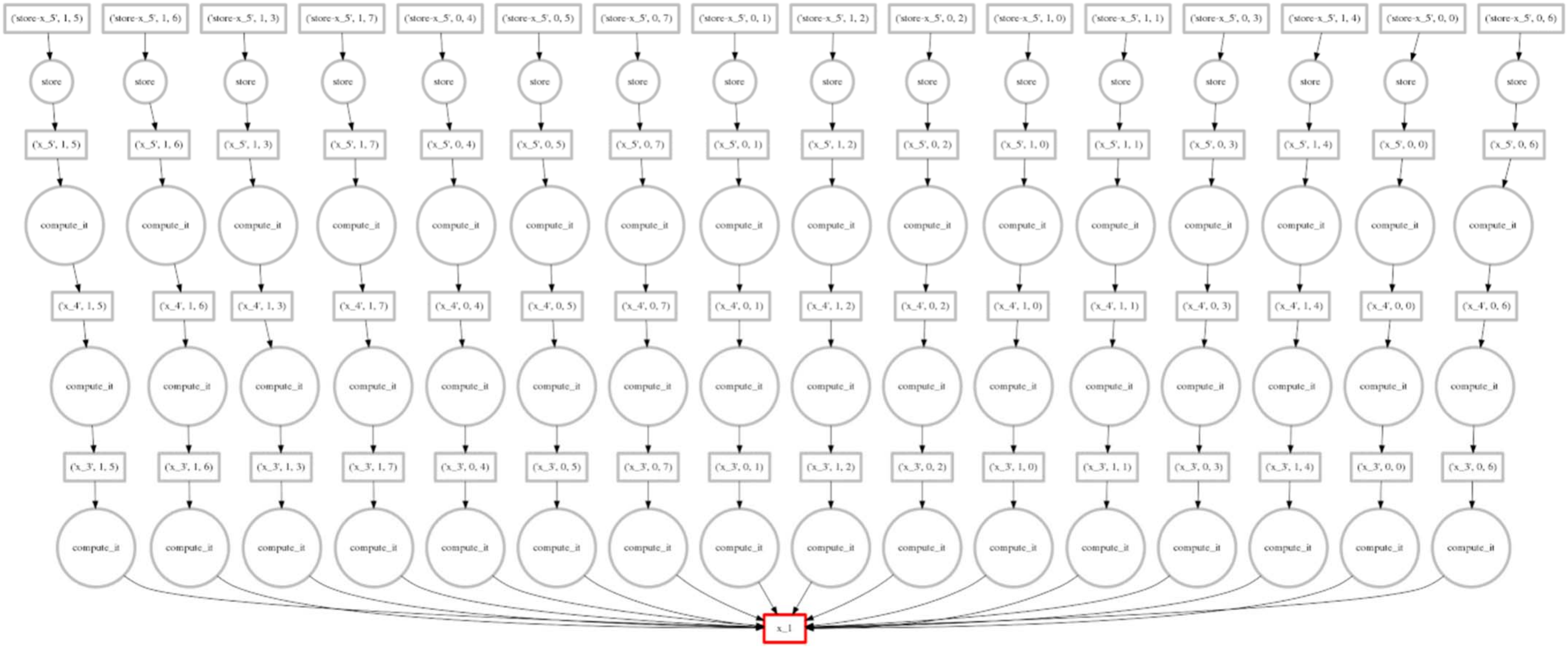
# Dask

To parallelize a code, a calculation graph is used.

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

def double(x):
   return x + 2

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

>>> x = dask.delayed(inc)(1)
>>> y = dask.delayed(inc)(2)
>>> z = dask.delayed(add)(x, y)
>>> z.compute()
5
>>> z.vizualize()
```



# Scaling the algorithm is automatic and simple with Dask



# Zarr

- File format designed for distributed computing
- Zarr is an interesting alternative to NetCDF4 for internal storage.
- Pure solution with transparent key-value storage.

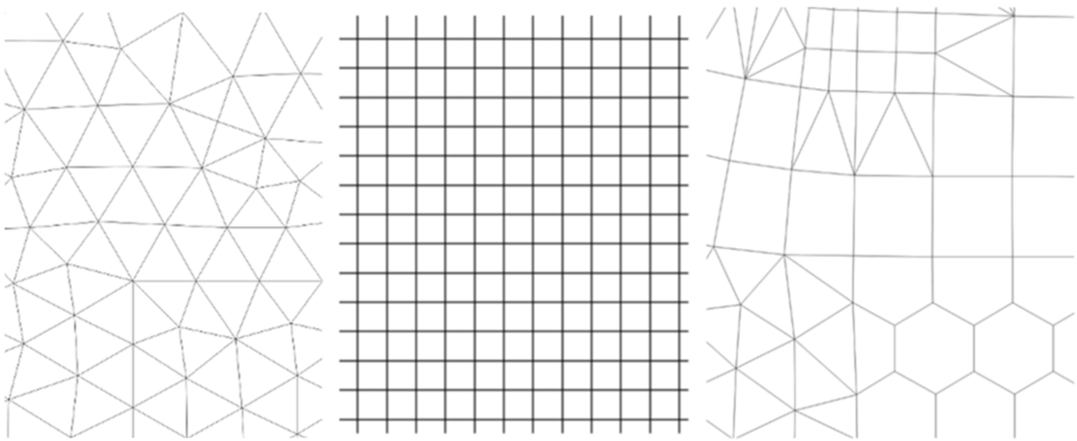
```
def load_faces(face, chunks):
    """Load a face from the time series"""
    ds=xarray.open_zarr(
        "/work/ALT/swot/Eta.zarr/")
    ds = ds.transpose("face", "j", "i", "time")
    return ds.isel(face=face, j=chunks, i=chunks)["Eta"].data

%time ds = load_faces(0, slice(0, None))
# Wall time: 1.65 s

%time ds.mean().compute()
# Wall time: 6min 28s (vs 1h 21mins with netCDF)
```

# Finally our algorithm becomes

```
for face in faces:
   ds = load_faces(face, chunks=var_chunk)
   ds = ds.rechunk(dask_array_rechunk(ds, 100))
   future = dask.apply_along_axis(
      tidal_constituents.WaveTable.harmonic_analysis,
      2,
      ds,
      (f, v0u))
   result = future.compute()
   result = numpy.transpose(result, [2, 0, 1])
   write_one_face(target, result, face, "Eta")
```

# Interpolate LLC4320 grids

Type of numerical grids



Triangular mesh　　Quad mesh (regular grid)　　Mesh with mixed element types

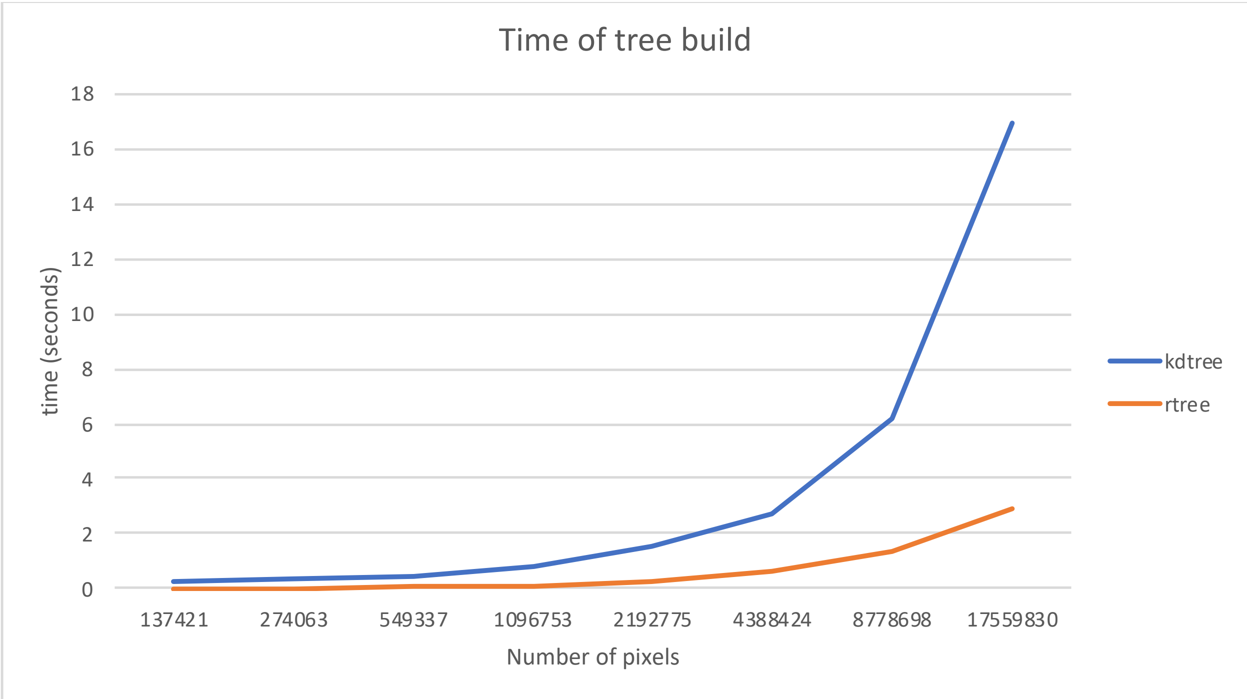
The earth is round!

# Existing tools

- There are many grid interpolation libraries, but in general they are written entirely with the standard Python stack and are not very efficient: MetPy, Verde, pyresample, etc.
- The basic tools are from numpy, scipy: RegularGridInterpolator, cKDTree.
- But none of these tools manages the discontinuity of longitudes: transition around the Prime meridian.

# A new library adapted to our problem.

- The algorithm is based on a search tree, very efficient whatever the size of our problem.
- The missing coordinate on the right in the graphs below is the total number of pixels of the MIT/GCM grids as the KDTree algorithm cannot handle it.



# Finally, to interpolate the grid LLC4320

```
# Creating the tree.
lon, lat, eta = load()
interpolator = core.interp2d.Mesh()
x, y = np.meshgrid(lon, lat, indexing='ij')
interpolator.packing(x.flatten(), y.flatten(), eta.flatten())
# Creation of the grid to be produced.
lon = np.arange(-180, 180, 1 / 3.0) + 1 / 3.0
lat = np.arange(-90, 90, 1 / 3.0) + 1 / 3.0
x, y = np.meshgrid(lon, lat, indexing="ij")
# And finally we quickly interpolate: 15 seconds of calculation on a PC.
values, samples = interpolator.inverse_distance_weighting(
   x.flatten(),
   y.flatten(),
   around=True,
   radius=18000,
   k=16,
   num_threads=0)
```