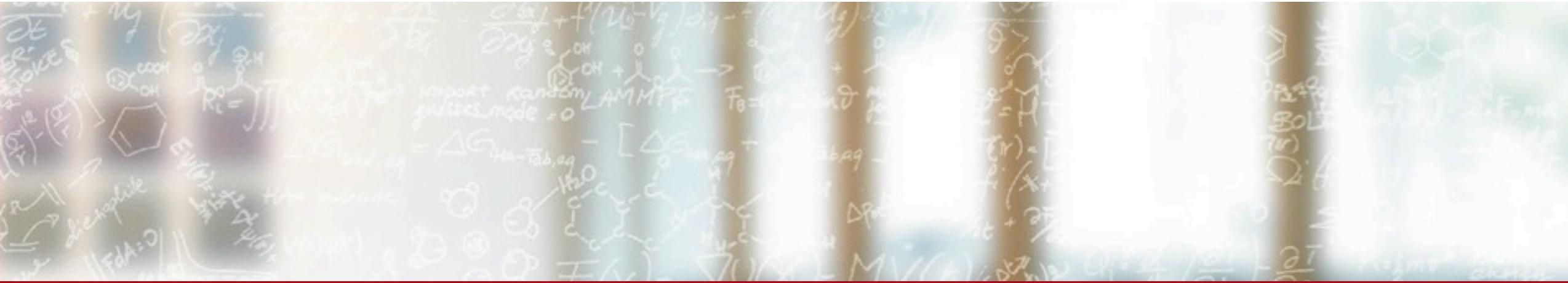




**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Interactive HPC on Piz Daint with JupyterLab

HPC-CH

Maxime Martinasso, CSCS

Oct 22, 2020

# Outline

1. Interactive supercomputing
  - Jupyter notebooks, JupyterHub, JupyterLab
2. Challenges in offering on-demand resources
  - Interactive vs batch scheduling
  - Technical solution
3. Jupyter features enabled on Piz Daint
  - Parallel computing notebooks
  - Custom environments
  - Integration improvements
4. Future plans



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

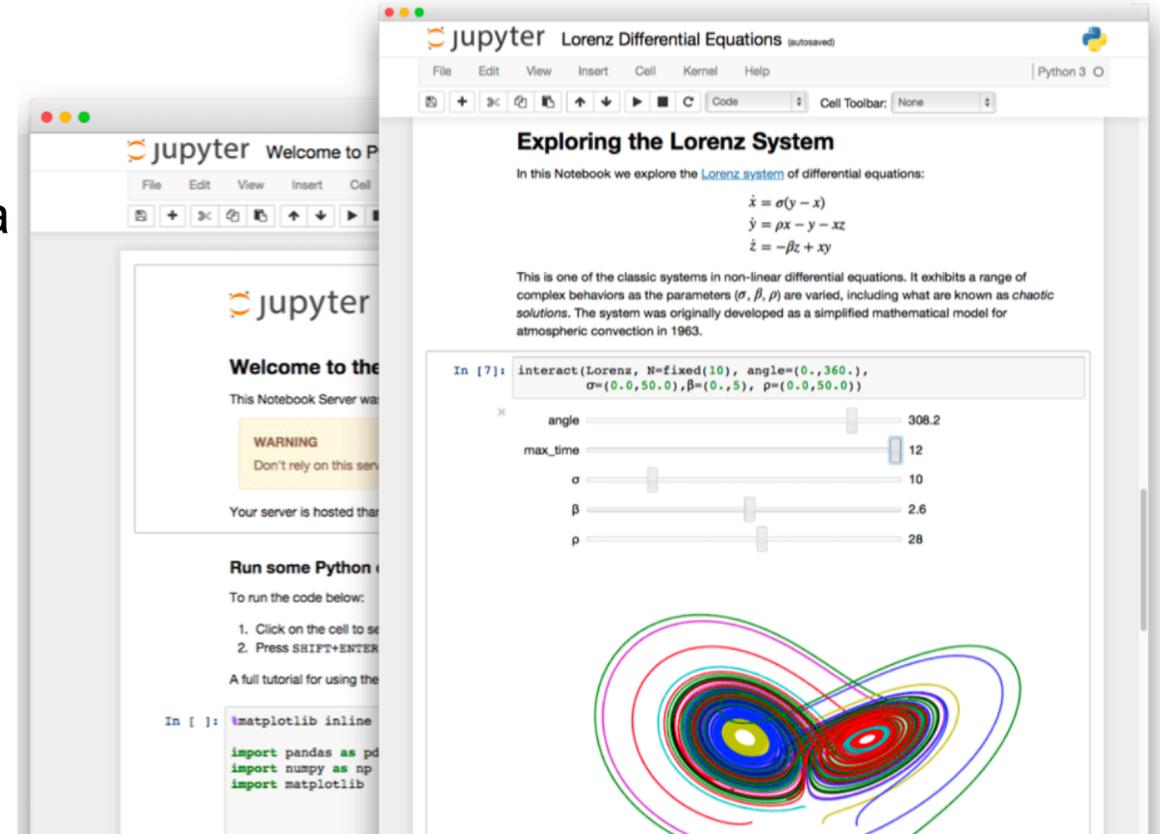
**ETH** zürich

# Interactive supercomputing

---

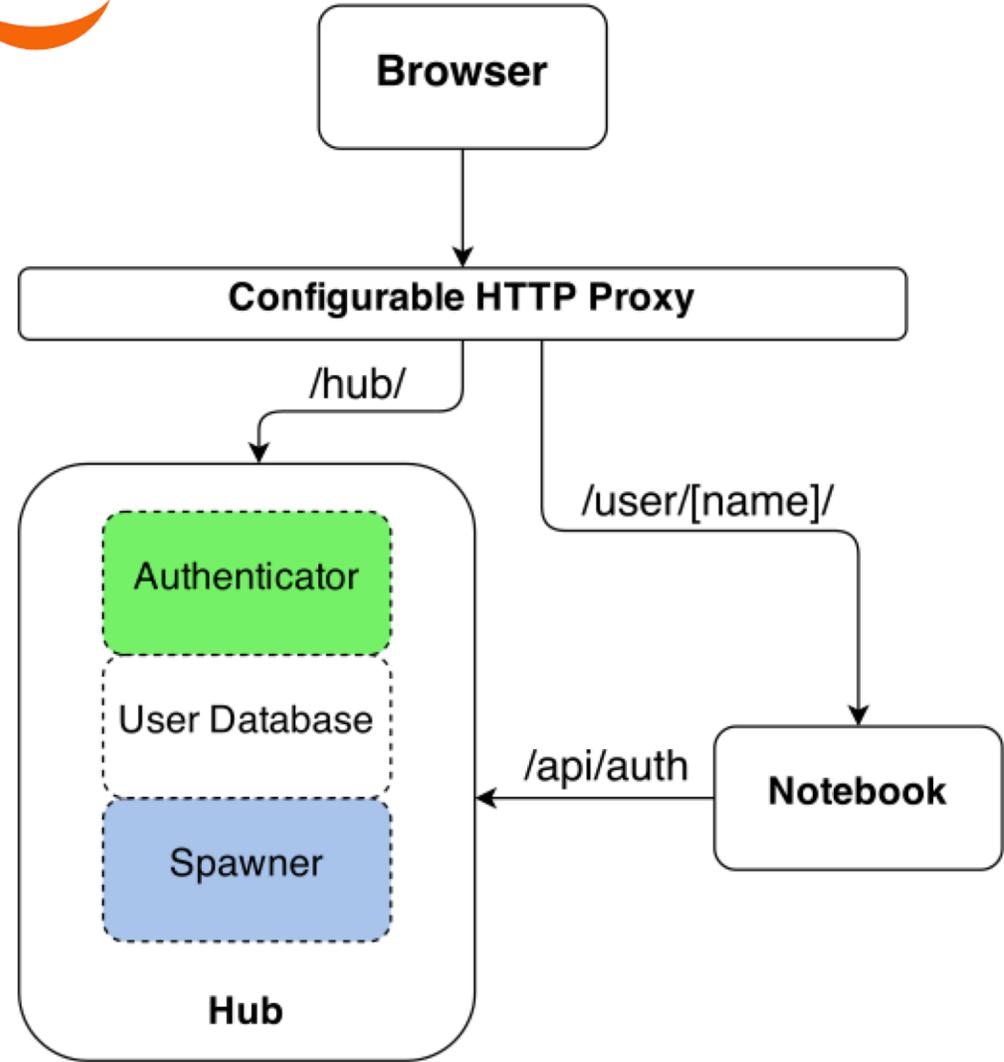
# Browser-enabled working environments

- Project Jupyter – enabling interactive computational environments in a web browser
- Jupyter is an open-source web application for creating **reproducible computational narratives**
- Create documents that contain live code, equations, narrative text, visualizations, rich media
- The all-in-one document is also “Jupyter Notebook”
  - easily shared with others
  - in-browser terminal
  - file browsing
  - Python, R, Julia, C++, ...
  - extensible design
  - many server/client plugins



# JupyterHub

- Multi-user server for Jupyter Notebooks (designed for classrooms, research labs, Universities...)
- Spawns, manages and proxies multiple instances of the single-user Jupyter Notebook server
- Three main subsystems
  - a **multi-user Hub** (tornado process)
  - a configurable **http proxy** (node-http-proxy)
  - multiple **single-user Jupyter notebook servers** (Python/IPython/tornado)
- The key pluggable components are the authenticator and spawner



# JupyterLab

File Edit View Run Kernel Tabs Settings Help

Files

- notebooks
- Data.ipynb (an hour ago)
- Fasta.ipynb (a day ago)
- Julia.ipynb (a day ago)
- Lorenz.ipynb (seconds ago)**
- R.ipynb (a day ago)
- iris.csv (a day ago)
- lightning.json (9 days ago)
- lorenz.py (3 minutes ago)

Running

Commands

Cell Tools

Output View

lorenz.py

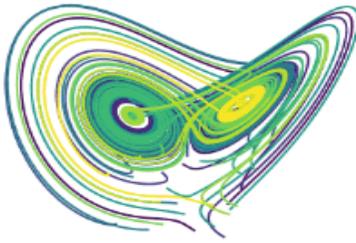
In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```
In [4]: from lorenz import solve_lorenz
t, x_t = solve_lorenz(N=10)
```

sigma 10.00  
beta 2.67  
rho 28.00



```
def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random.random((N, 3))
```



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

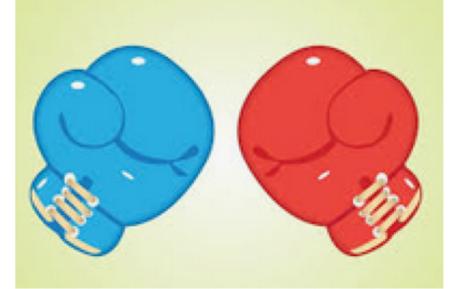
**ETH** zürich

# Challenges in offering interactivity on HPC

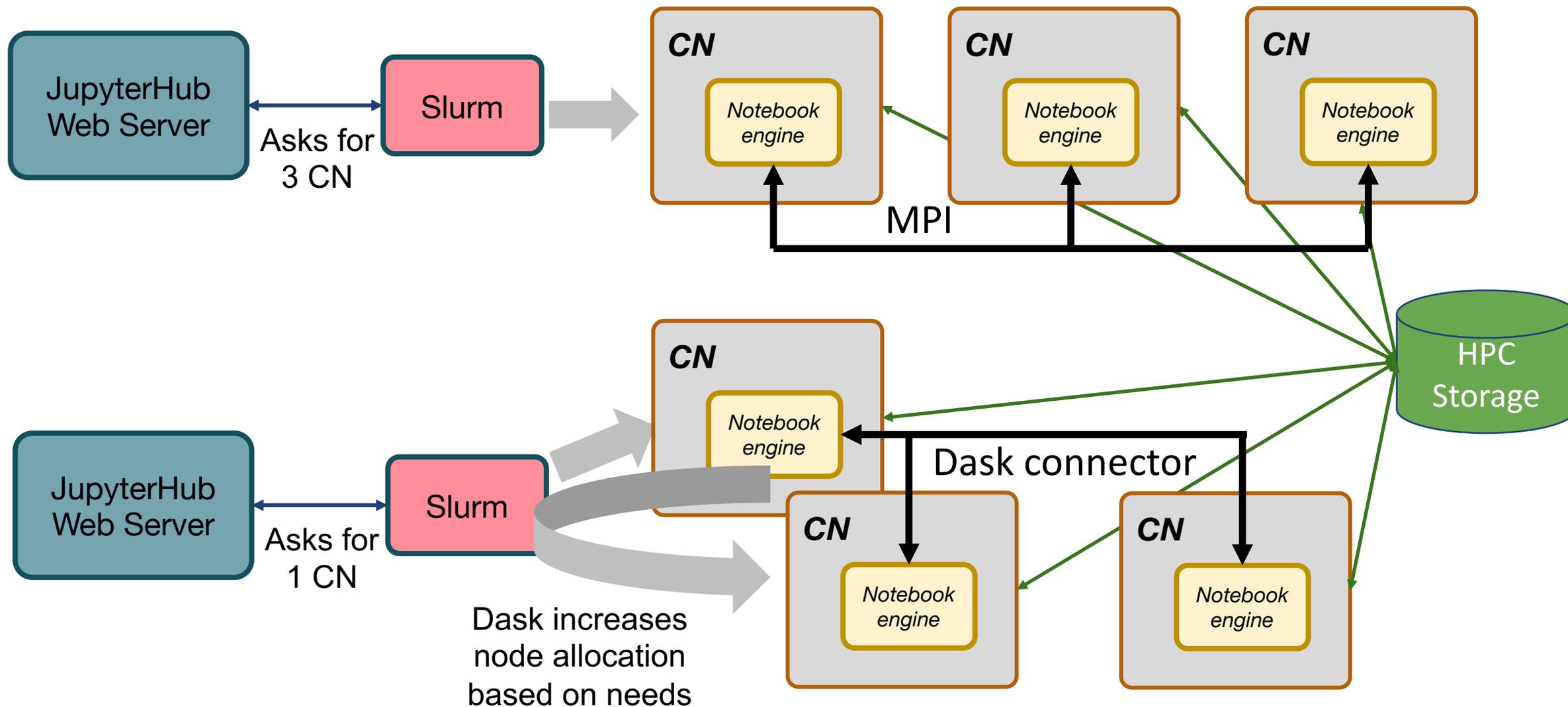
---

# Challenge: Batch vs interactive computing

- How can we reconcile the apparent contradiction between **batch computing** and **interactive computing**?
- Batch is used on HPC to maximize utilization, large jobs
- Interactivity = small waiting time of jobs in queue
- Job pre-emption? Suspend/Resume? Reservation?
  
- Our solution: Elastic reservation
  - Increase/decrease number of nodes depending on the demand
  - Minimum fix number of nodes
  - Slurm flag REPLACE

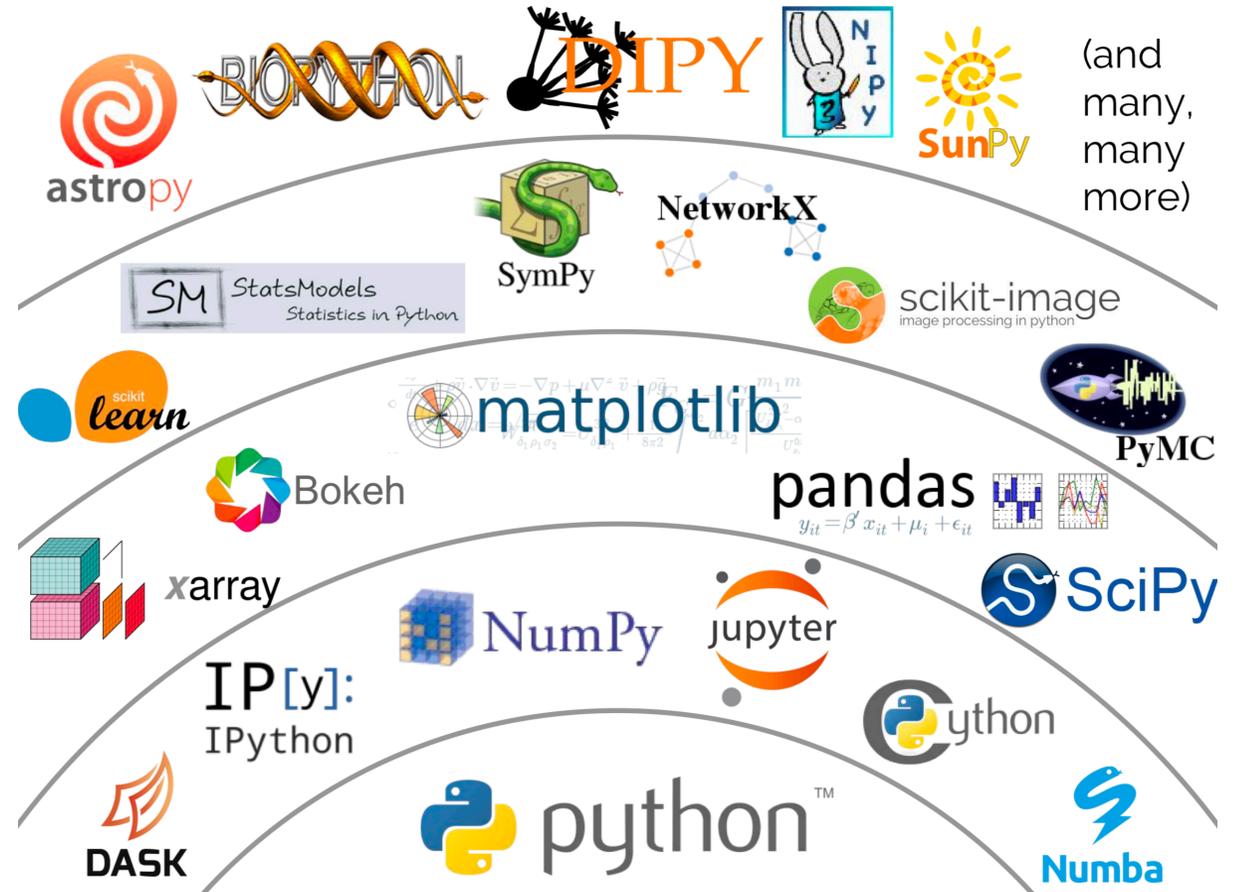


# Static and dynamic resource allocations

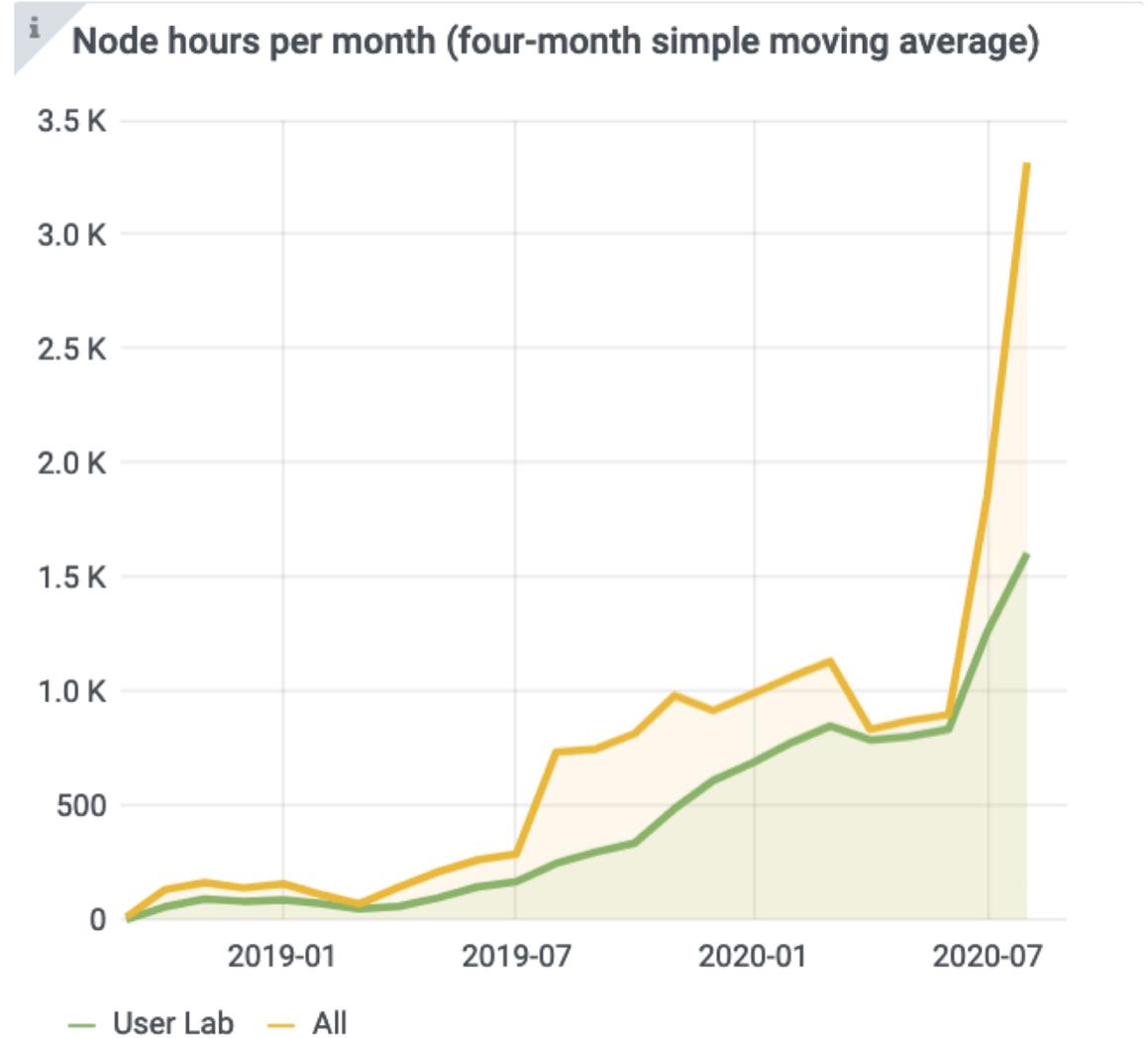
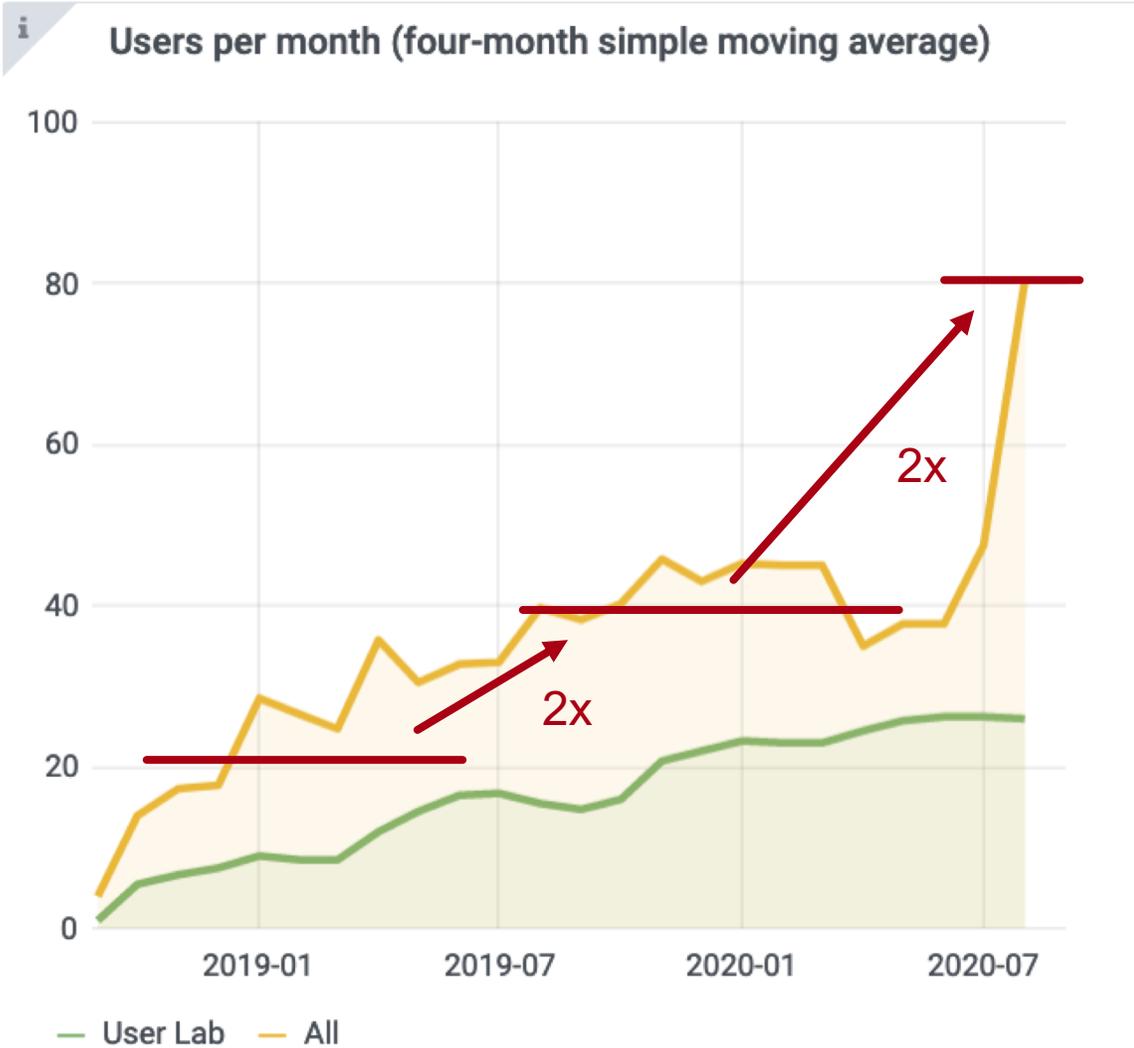


# Jupyter software stack at CSCS

- Carefully manage package versions
  - Installed with EasyBuild
- Based on `cray-python/3.x`
  - provides `numpy` and `scipy` that call `cray-libsci` routines
- Parallel computing available in the notebook
  - `ipyparallel` (MPI with `mpi4py`)
  - distributed `dask`



# JupyterLab usage over time





**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Jupyter features on Piz Daint

---

# Submission options

The screenshot shows the JupyterHub submission page for CSCS at ETH Zürich. The page is titled "JupyterHub" and the URL is "https://jupyter.cscs.ch/hub/spawn". The user is identified as "robinson". The main form contains the following fields and options:

- Node Type:** A dropdown menu set to "GPU".
- Nodes:** A numeric input field set to "1".
- Duration (hr):** A numeric input field set to "1".
- Queue:** A dropdown menu set to "Dedicated Queue (Max. 4 Nodes)".
- Project Id (leave empty for default):** An empty text input field.
- Advanced Reservation:** An empty text input field.
- JupyterLab Version:** A dropdown menu set to "1.1.1".
- Start IPyParallel Cluster with MPI Support?:** Radio buttons for "No" (selected) and "Yes".
- MPI Processes Per Node (default: one process per virtual core):** A numeric input field set to "1".
- Start Distributed Dask Cluster?:** Radio buttons for "No" (selected) and "Yes".
- Dask Tasks Per Node (default: one task per node):** A numeric input field set to "1".

At the bottom of the page, there are links for "Help | Privacy | Terms" and a copyright notice: "2019 © CSCS | www.cscs.ch".

Queue

Reservation

IPyParallel  
(multi-node  
notebooks)

Distributed  
Dask cluster  
(multi-node  
notebooks)

Project Id  
(account)

JupyterLab  
version

# Launching notebook

Memory monitor

The screenshot displays the JupyterLab interface. On the left, the 'Server console' shows a message: 'Your server is starting up. You will be redirected automatically when it's ready for you.' Below this, it states 'Pending in queue...The job is waiting for resources to become available.' An 'Event log' section contains a list of 'Pending in queue...' messages. The central file browser shows a directory listing with columns for 'Name' and 'Last Modified'. The right panel, titled 'Launcher', offers options for 'Notebook' (Python 3, Bash, Julia 1.5.0, miniconda-pythonhpc, paraview) and 'Console' (Python 3, Bash, Julia 1.5.0, miniconda-pythonhpc, paraview). At the bottom of the launcher, there are 'Other' options: Terminal, Text File, Markdown File, and Show Contextual Help. The top right corner of the browser window shows 'Mem:232 / 62464 MB'.

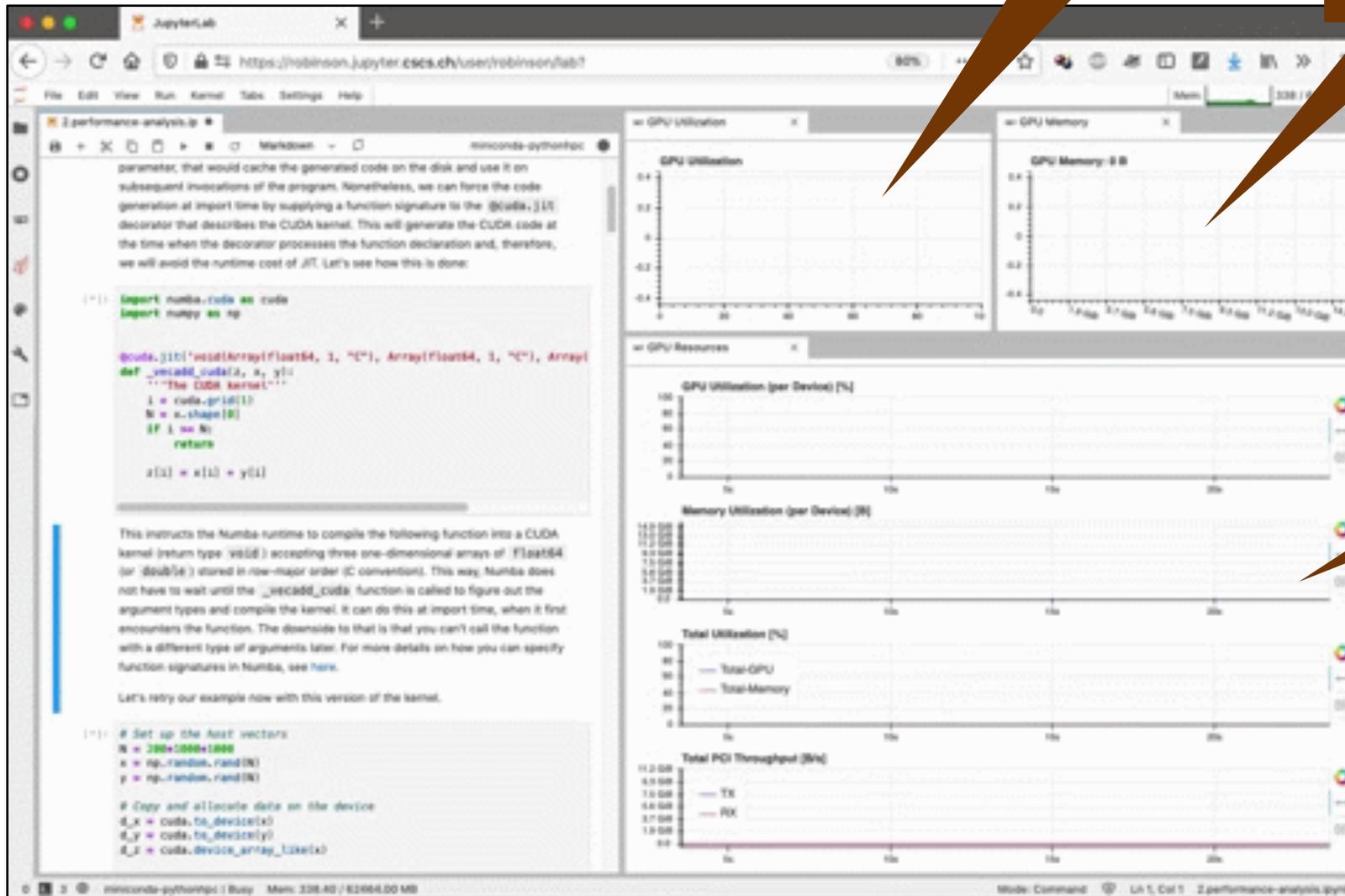
Name	Last Modified
aws	3 months ago
awscliv2	3 months ago
bin	2 months ago
cdt2006	a month ago
dask-worker...	7 days ago
Documents	2 years ago
easybuild_files	2 months ago
gt4py	6 days ago
holoviews	22 days ago
hpc_python_...	a year ago
include	7 years ago
ipyparaview	5 days ago
lib	6 years ago
lib64	5 years ago
logs	a month ago
metastore_db	2 years ago
multiprocessi...	a year ago
node_modules	24 days ago
notebooks	a day ago
paraview	6 days ago
production	25 days ago
project	a month ago
public_html	12 years ago
python	4 hours ago
Python_NIWA...	a month ago

Event log

# GPU dashboards

Compute

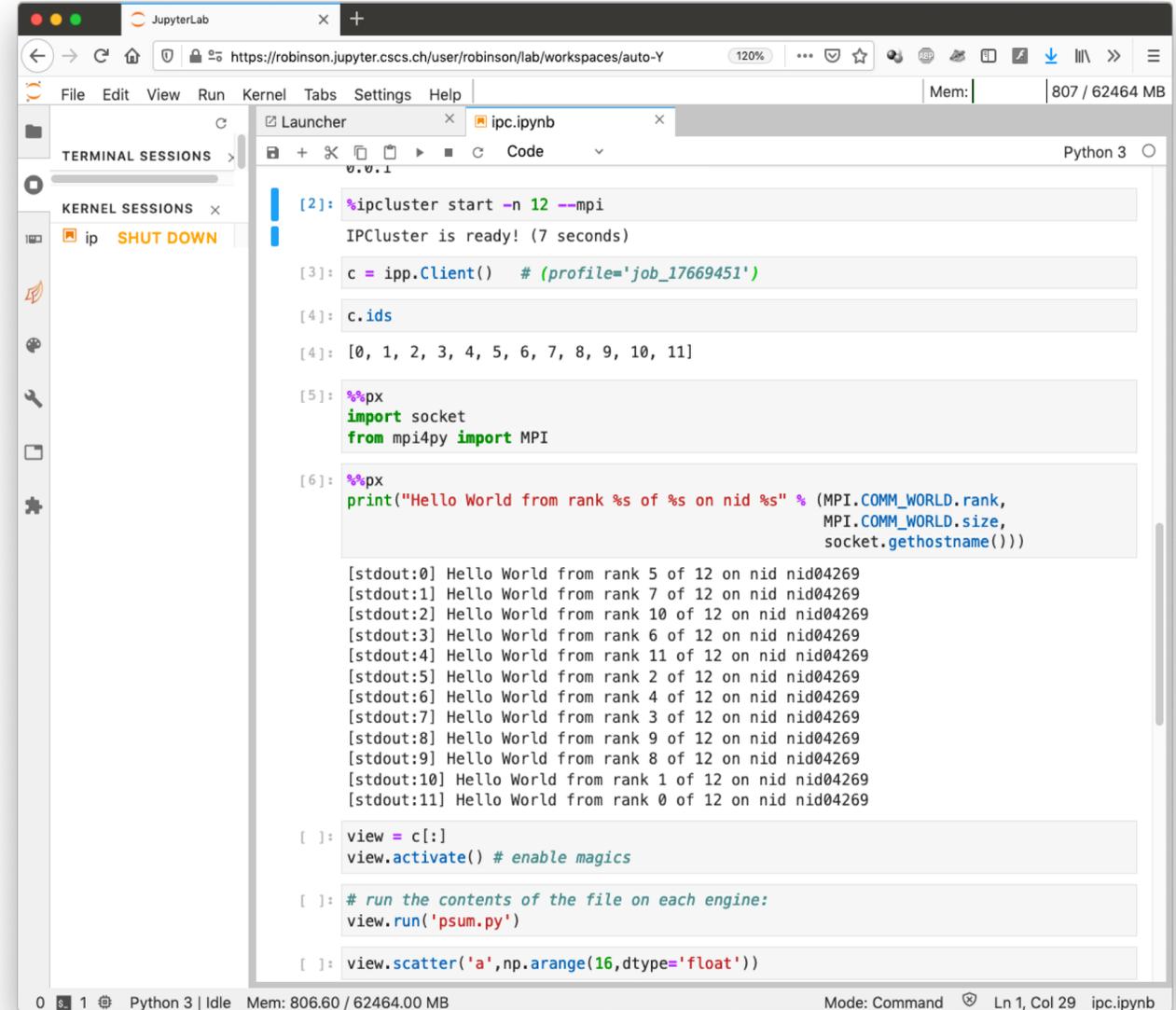
Memory



Timeline:  
- Compute  
- Memory  
- Total utilization  
- PCIe throughput

# IPCluster magic

- IPCluster magic commands for automatically starting and stopping ipyparallel clusters (ipcontroller and ipengines)
- Enables the use of MPI directly in the notebook
- Developed at CSCS, inspired by NERSC's `%ipcluster_magics`



The screenshot shows a JupyterLab interface with a notebook titled 'ipc.ipynb'. The notebook contains the following code cells:

```
[2]: %ipcluster start -n 12 --mpi
IPCluster is ready! (7 seconds)

[3]: c = ipp.Client() # (profile='job_17669451')

[4]: c.ids
[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

[5]: %%px
import socket
from mpi4py import MPI

[6]: %%px
print("Hello World from rank %s of %s on nid %s" % (MPI.COMM_WORLD.rank,
MPI.COMM_WORLD.size,
socket.gethostname()))

[stdout:0] Hello World from rank 5 of 12 on nid nid04269
[stdout:1] Hello World from rank 7 of 12 on nid nid04269
[stdout:2] Hello World from rank 10 of 12 on nid nid04269
[stdout:3] Hello World from rank 6 of 12 on nid nid04269
[stdout:4] Hello World from rank 11 of 12 on nid nid04269
[stdout:5] Hello World from rank 2 of 12 on nid nid04269
[stdout:6] Hello World from rank 4 of 12 on nid nid04269
[stdout:7] Hello World from rank 3 of 12 on nid nid04269
[stdout:8] Hello World from rank 9 of 12 on nid nid04269
[stdout:9] Hello World from rank 8 of 12 on nid nid04269
[stdout:10] Hello World from rank 1 of 12 on nid nid04269
[stdout:11] Hello World from rank 0 of 12 on nid nid04269

[ ]: view = c[:]
view.activate() # enable magics

[ ]: # run the contents of the file on each engine:
view.run('psum.py')

[ ]: view.scatter('a', np.arange(16, dtype='float'))
```

The notebook interface includes a terminal on the left showing 'ip SHUT DOWN', a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Tabs', 'Settings', 'Help', and a status bar at the bottom indicating 'Python 3 | Idle', 'Mem: 806.60 / 62464.00 MB', 'Mode: Command', and 'Ln 1, Col 29 ipc.ipynb'.

# Creating and installing a custom kernel

```
robinson@dain106:~> module load cray-python
robinson@dain106:~> module load daint-gpu # or module load daint-mc
robinson@dain106:~> module load jupyter-utils
robinson@dain106:~> python -m venv daint-xarray
robinson@dain106:~> source daint-xarray/bin/activate
(daint-xarray) robinson@dain106:~> pip install xarray
```

1

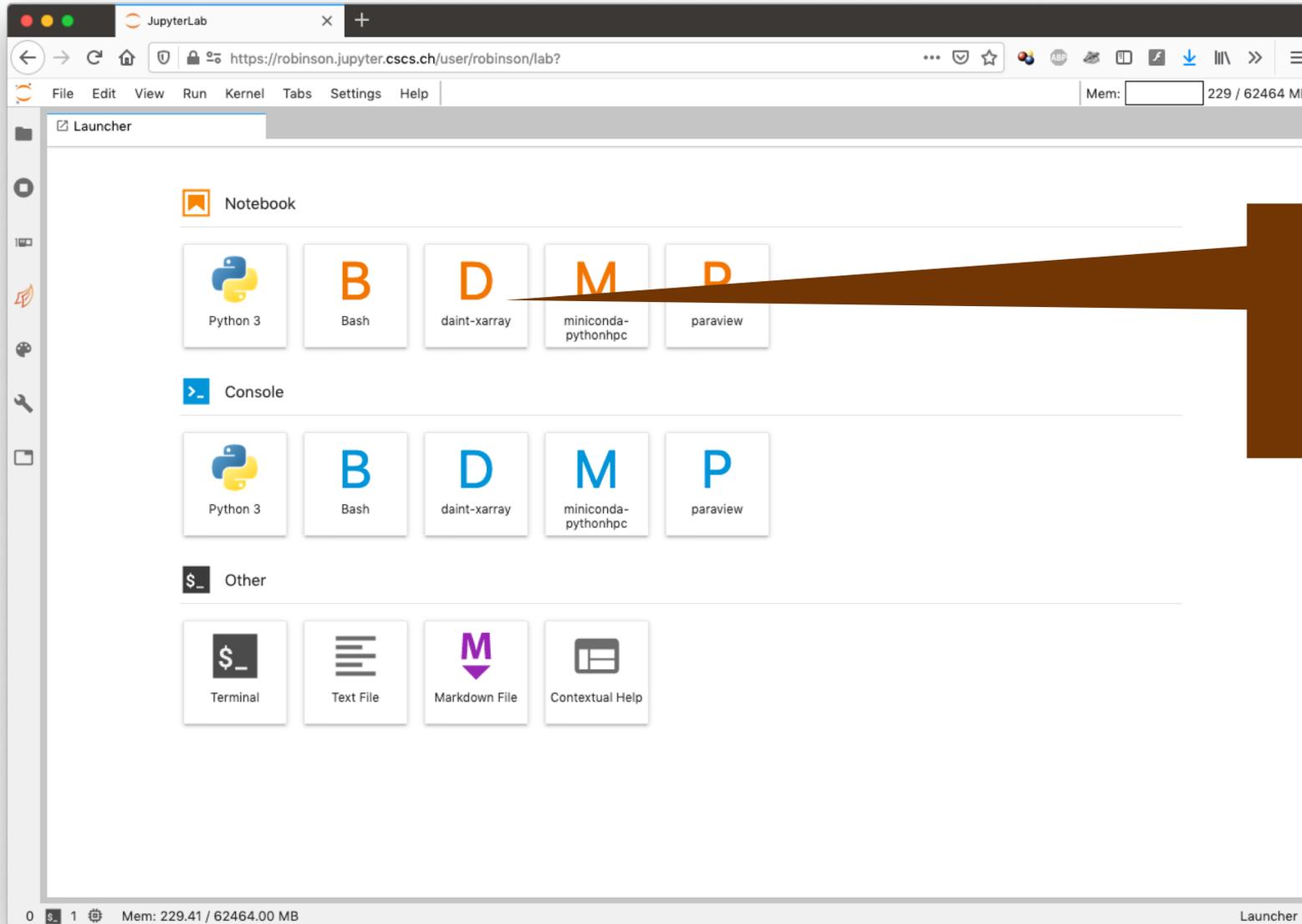
Create a virtualenv with your packages

```
(daint-xarray) robinson@dain106:~> kernel-create -n daint-xarray
Kernel 'daint-xarray' created successfully in '/users/robinson/.local/share/jupyter/kernels/daint-xarray'
(daint-xarray) robinson@dain106:~>
```

2

Create a JupyterLab kernel from the virtualenv

# Using the newly created kernel



daint-xarray kernel will appear immediately

# Integration improvement

- IPyParaview is an iPython widget for server-side ParaView rendering in Jupyter

- Adding support for Julia notebook



cscs RayTracing Last Checkpoint: 2 minutes ago (unsaved changes) Logout Control Panel

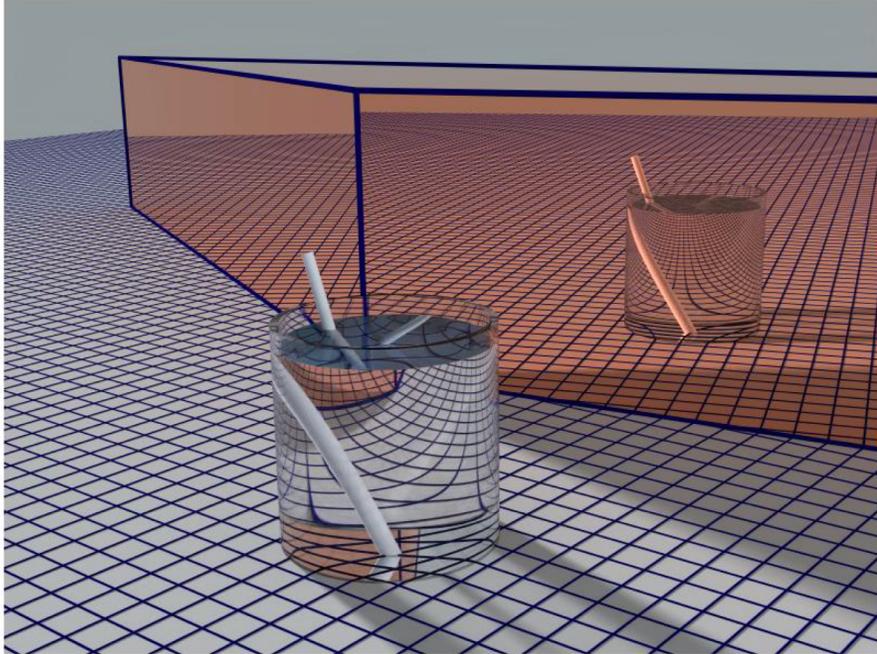
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
renderView1.AdditionalLights = [light1, light2]
renderView1.OSPRayMaterialLibrary = materialLibrary1
```

In [6]: `renderView1.ViewSize = [800, 600]`

In [7]: `# import the PVDdisplay widget, then instantiate one for renv
from ipyparaview.widgets import PVDdisplay
disp = PVDdisplay(renderView1)

# show the widget once
display(disp)`



In [8]: `from paraview.modules.vtkPVClientServerCoreRendering import vtkPVOpenGLInformation

info = vtkPVOpenGLInformation()
info.CopyFromObject(None)
print("Vendor: %s" % info.GetVendor())
print("Version: %s" % info.GetVersion())
print("Renderer: %s" % info.GetRenderer())`

Vendor: NVIDIA Corporation  
Version: 4.6.0 NVIDIA 418.39  
Renderer: Tesla P100-PCIE-16GB/PCIe/SSE2

# JupyterHub announcement service

- We need to communicate with our users about status, upcoming outages, or share news
- Adding an announcement required system administrator intervention, and a restart of the JupyterHub server, which killed all user's notebook servers!
- JupyterHub recently added support for "services"
- We deployed an announcement service that is managed via a user interface
- Announcements can be added by user support staff directly
- No longer requires a restart of the JupyterHub server

Message is displayed to users without restarting server

JupyterHub

ETH zürich

Home Token Admin Services

User: robinson

**Announcement**

Jupyter service will be under maintenance this Friday

Node Type	Nodes	Duration (hr)
GPU	- 1 +	- 8 +

Advanced options

Launch JupyterLab

Help | Privacy | Terms

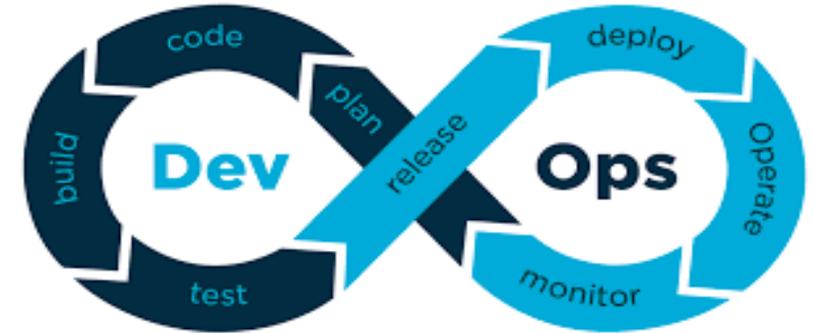
2019 © CSCS | www.cscs.ch

# Future plans

---

# Automatization of deployment

- Integrate Jupyter into the SRE process being developed
- Containerization of the Jupyter stack
- Automatic deployment of JupyterHub
- Environment configuration and management
- Improved unit testing and use case testing
- Better integration in IAM (for federation)



# Next features in JupyterLab

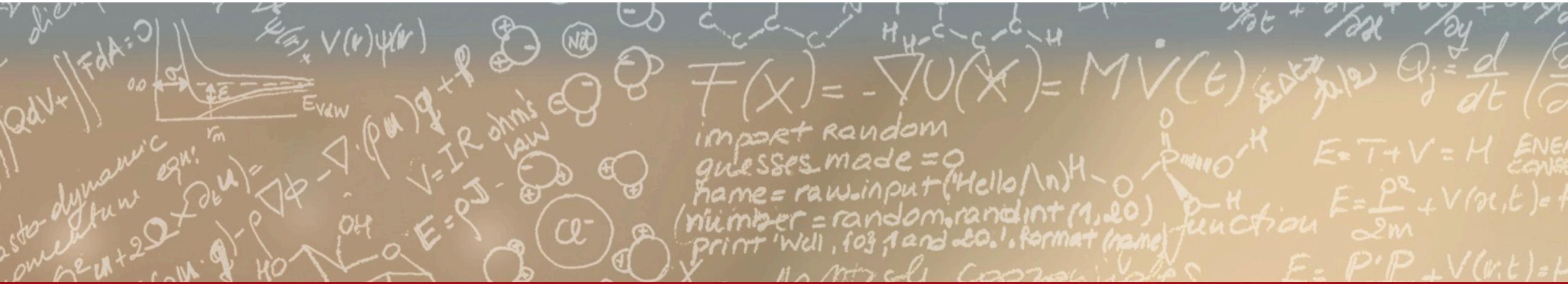
- User-defined JupyterLab stack
  - Containerized JupyterLab stacks can help reproducibility
  - Stacks be customized by users (e.g. installing specific lab extensions)
- Sharing notebook with a link on Piz Daint
  - Make notebooks accessible on-demand from a link, exchange/reproducibility
  - Useful to move notebook close to data generated by an HPC simulation
  - Binder-like service for HPC
  - GitHub repo + requirements.txt/environment.yml → automatic deployment of a notebook
  - Container + Kubernetes automatic deployment
- Zero to JupyterHub
  - Instantiate a JupyterHub infrastructure on-demand
  - Uses Kubernetes and containers
  - JupyterHub as a Service



CSCS

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

ETH zürich



Thanks in particular to Tim Robinson