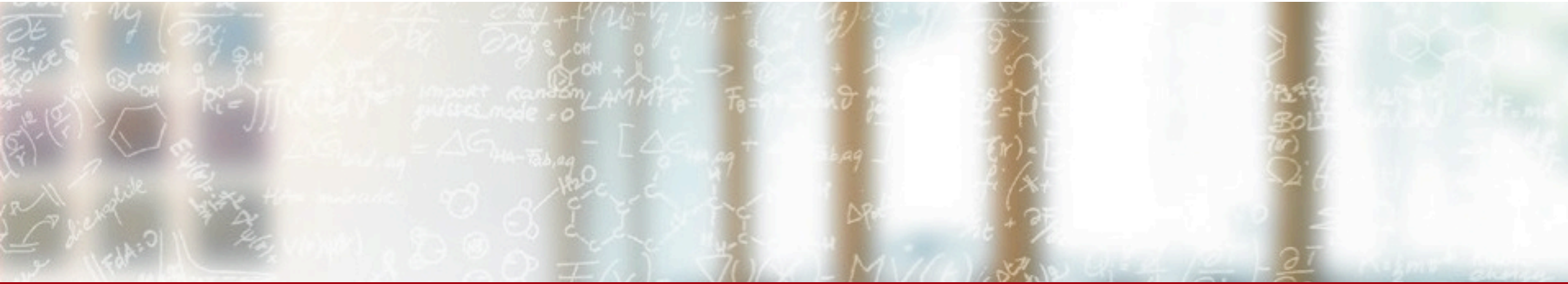




CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Challenges in Providing an Interactive Service with Jupyter on Large-Scale HPC Systems

CUG 2019, Montreal
Tim Robinson, CSCS
May 7, 2019

Outline

1. Interactive supercomputing

- Jupyter, JupyterLab, JupyterHub...

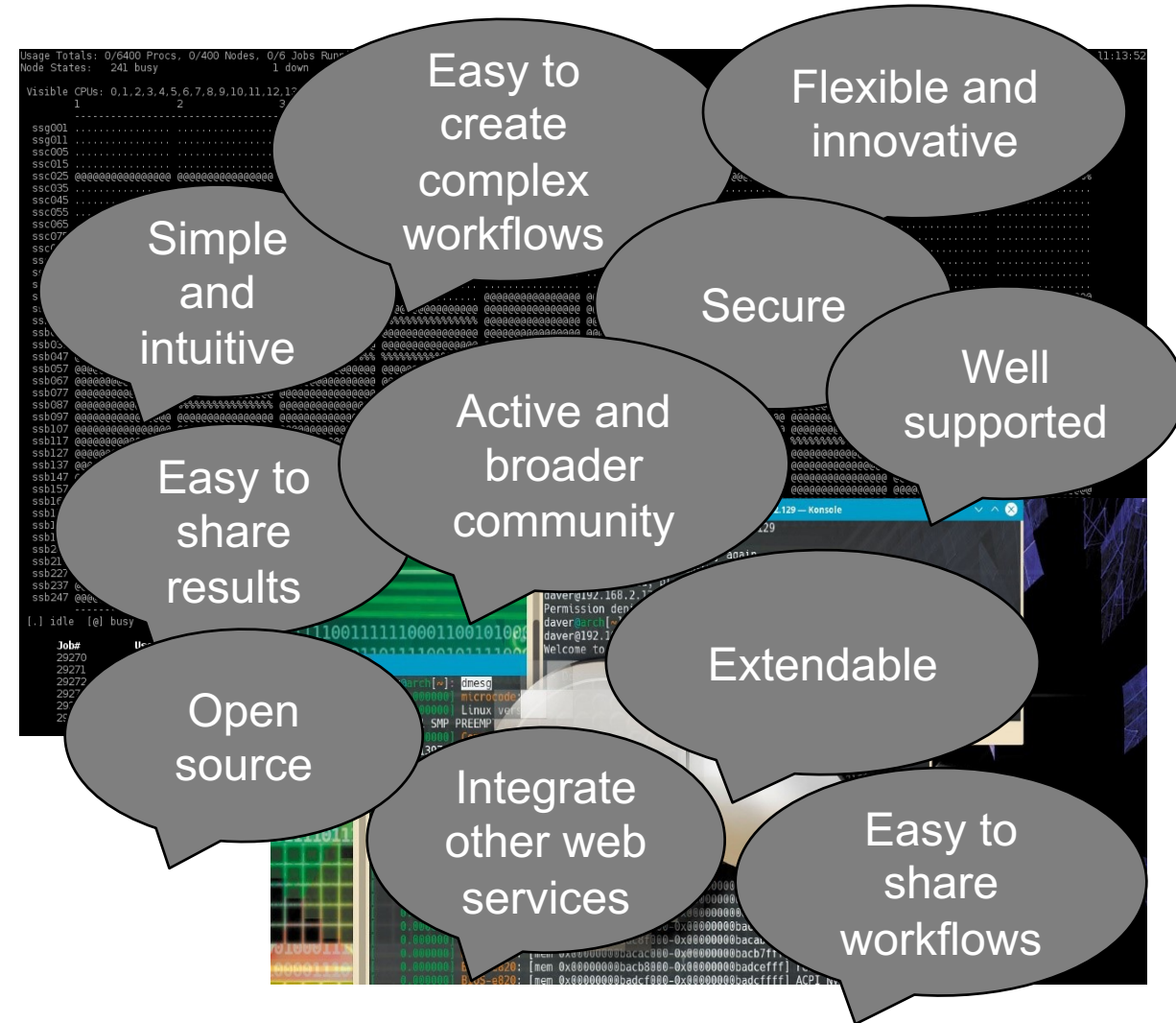
2. State of the practice at HPC centres

- Parallel computing (MPI, Dask...) – use cases
- Virtual environments and kernels
- JupyterLab extensions

3. Future plans

Classical supercomputing vs interactive supercomputing

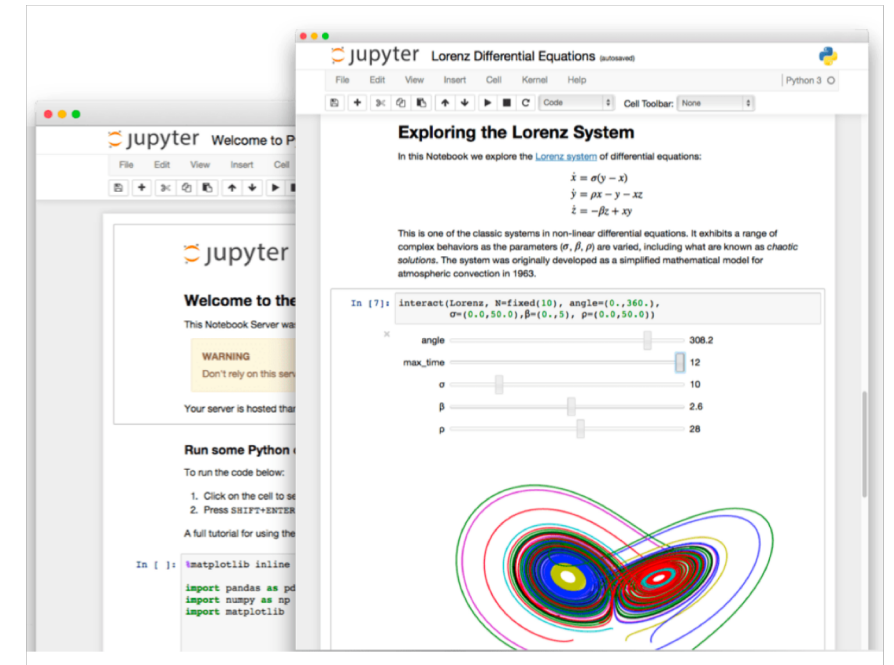
- Classical supercomputing would like to see...
 - batch operation
 - long-running jobs
 - terminal access
- But solutions to scientific problems often require an...
 - iterative,
 - interactive,
 - collaborative approach
- What is our wish list for providing iterative, interactive and collaborative supercomputing?
- Access to our supercomputers should be...



Slide credit: Jens Henrik Göbbert, Forschungszentrum Jülich

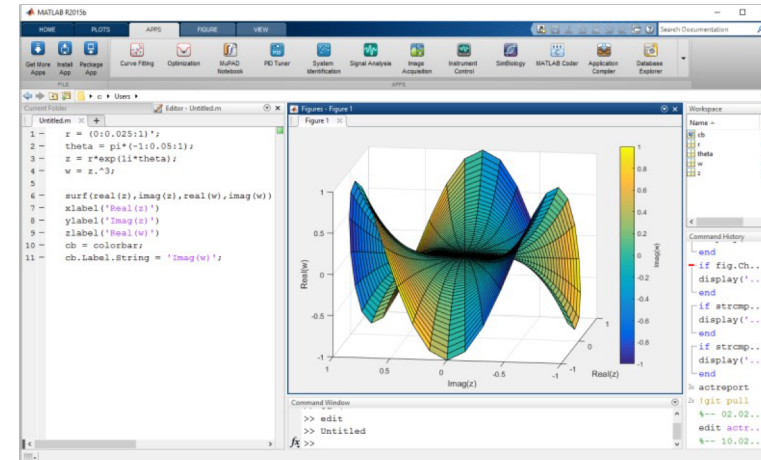
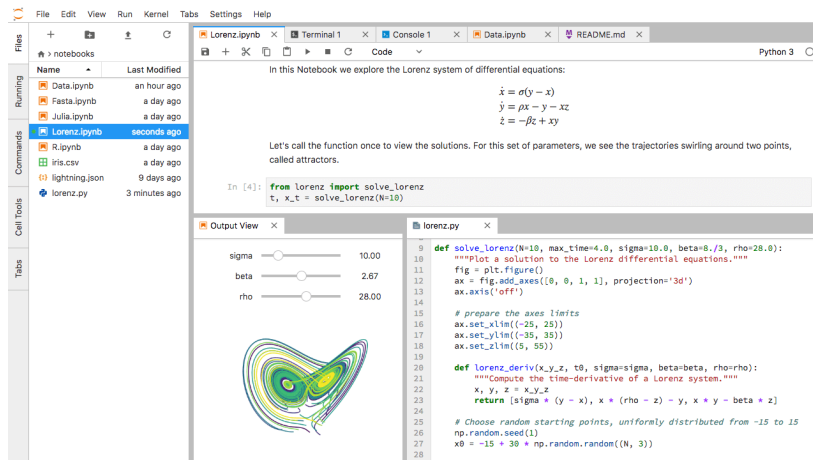
Browser-enabled working environments

- Project Jupyter – enabling interactive computational environments in a web browser
- Jupyter Notebook 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” (.ipynb, JSON format)
 - easily shared with others
 - convert to PDF, HTML, LaTeX
- The working environment includes
 - in-browser terminal
 - file browsing
 - support for many languages: Python, R, Julia, C++, ...
 - extensible design
 - many server/client plugins



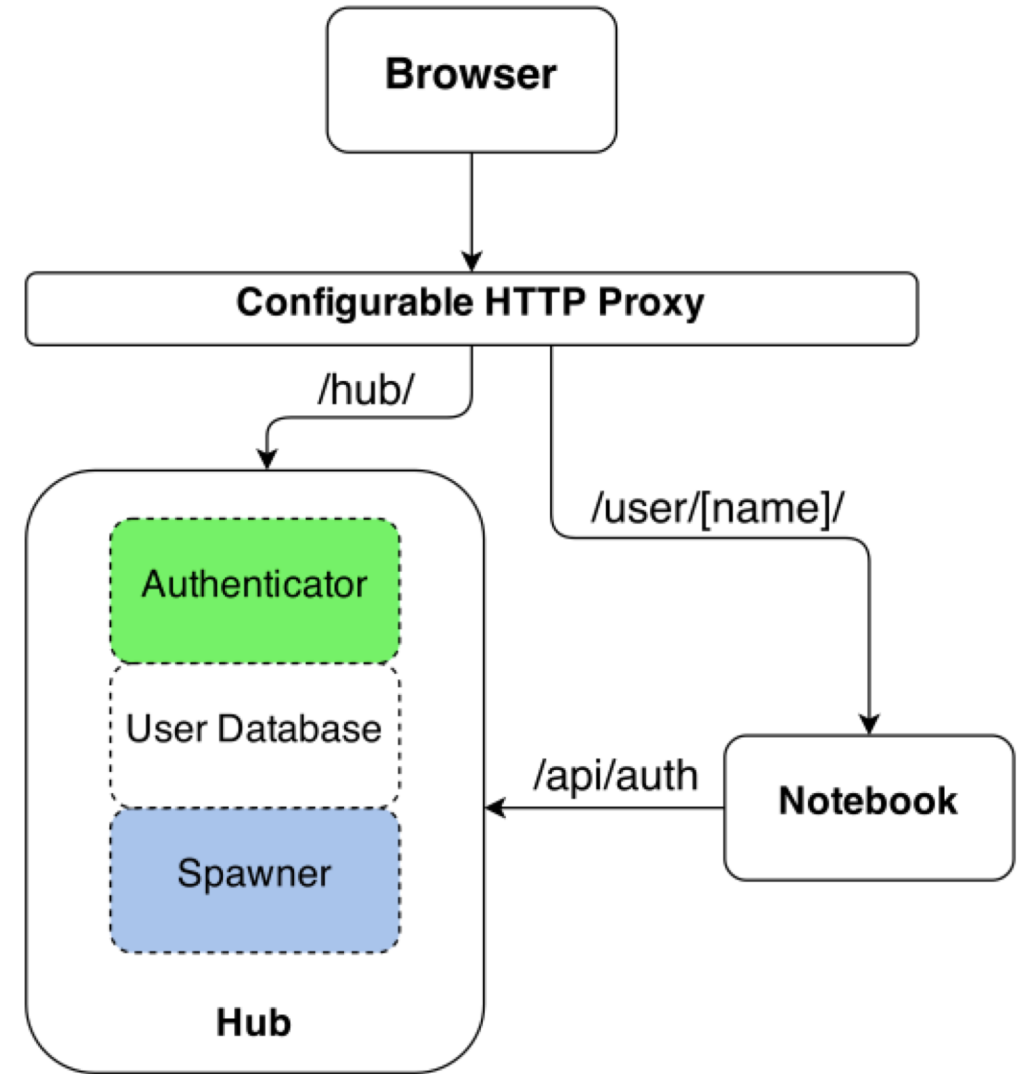
JupyterLab

- Next-gen web-based user interface for Jupyter
- Provides higher degree of interaction between notebooks, documents, text editors and other activities (arrange with tabs/splitters)
- Advanced interactive development environment
- Served from same server and uses same notebook document format



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



JupyterHub usage at CSCS

Spawner Options

Piz Daint node type

Queue

Training course reservation

Number of nodes

Job duration

Account (leave empty for default)

Start IPyParallel automatically with MPI?

If yes, how many processes per node? (default: one process per virtual core)

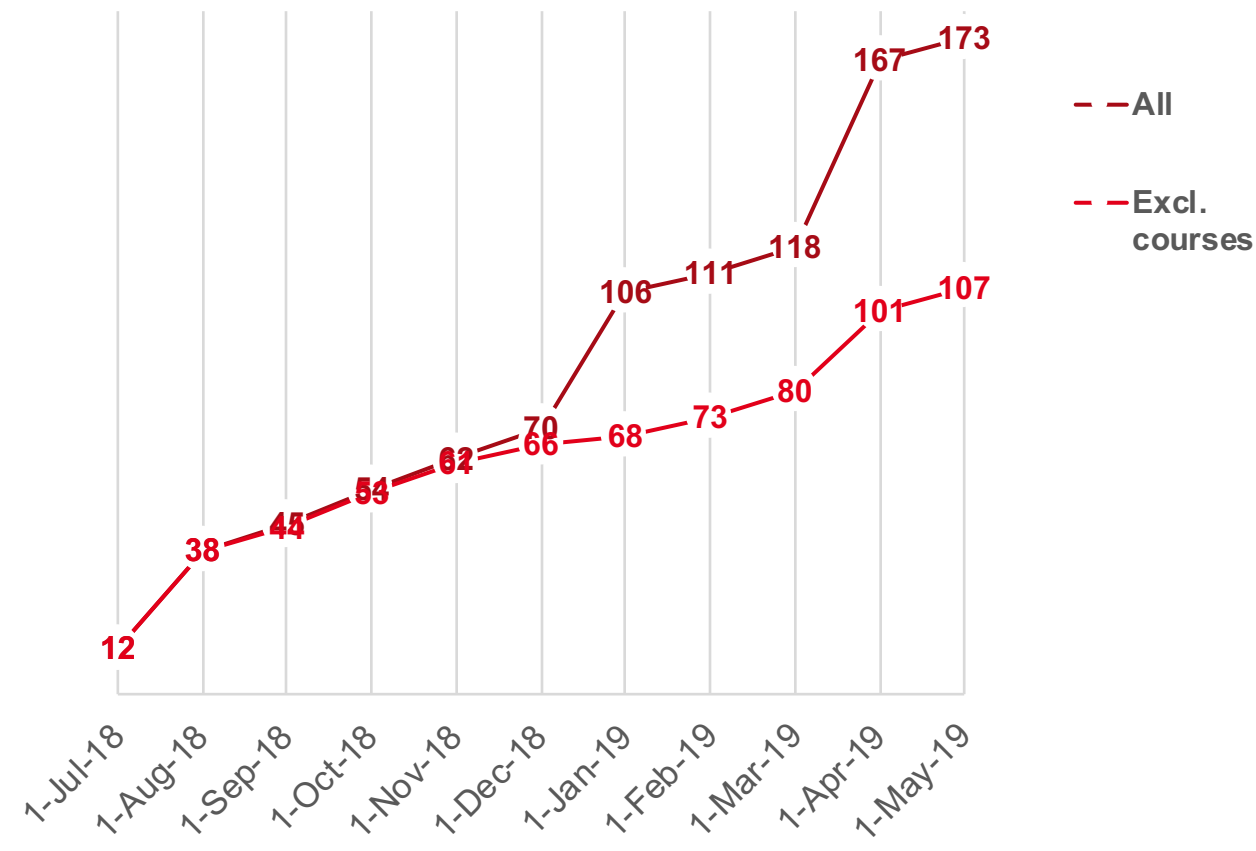
Start Dask.distributed automatically?

If yes, how many tasks per node? (default: one task per node)

NB: the number of threads = ncores / nprocesses

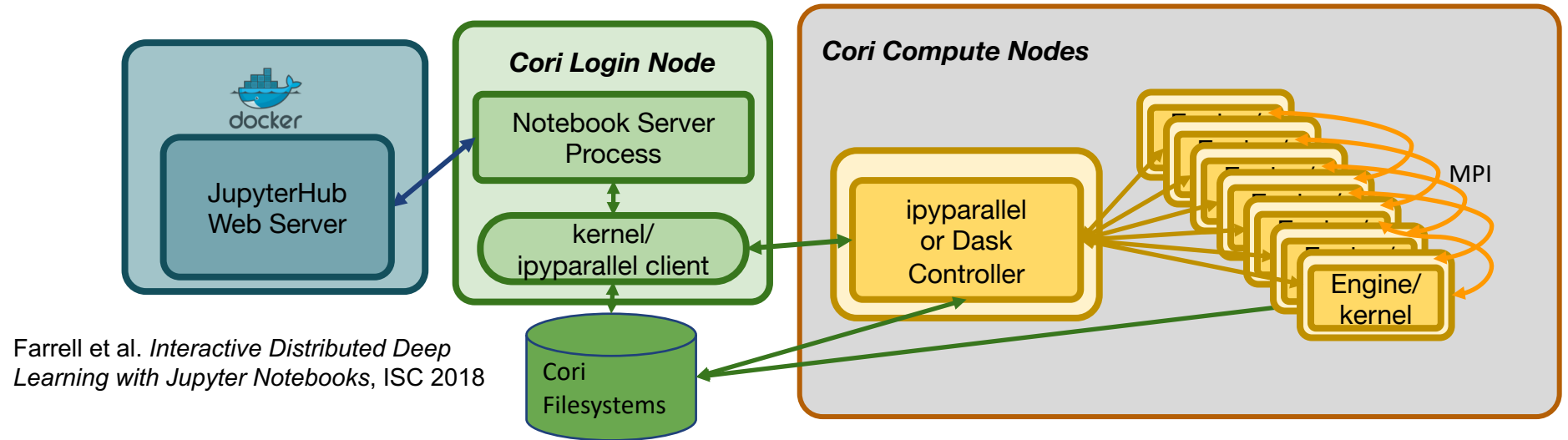
Spawn

UNIQUE USERS (RECORDED)

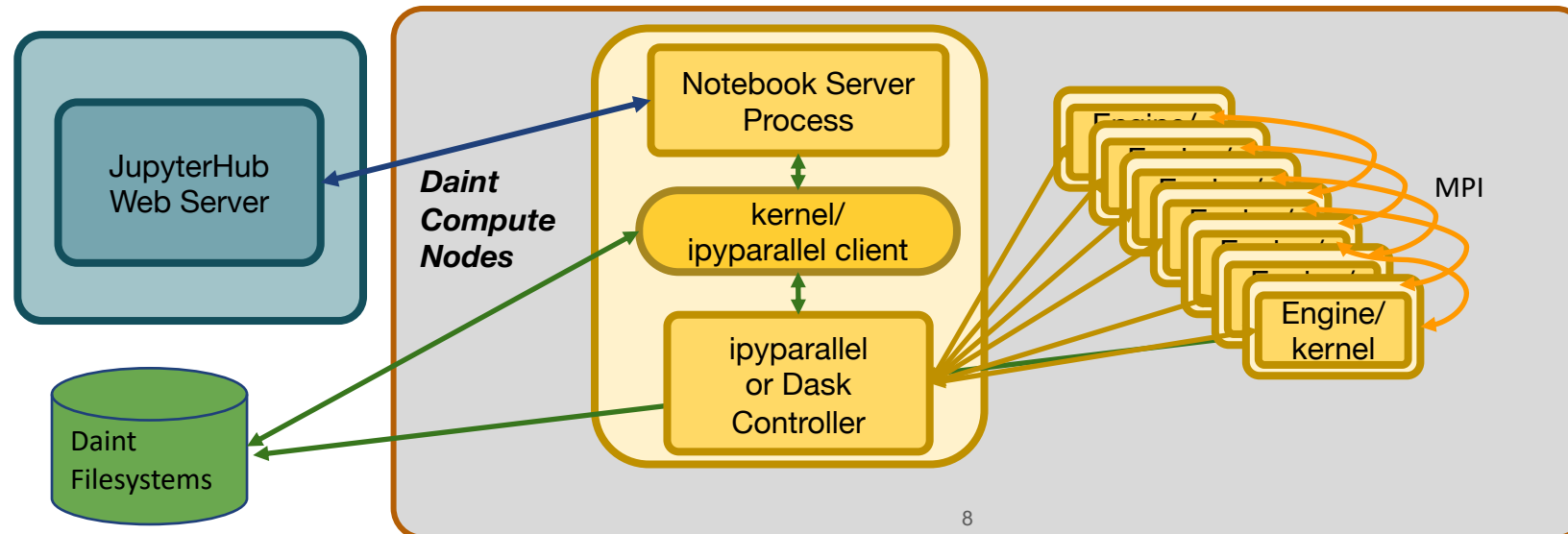


Current implementations of JupyterHub

- e.g. NERSC



- e.g. CSCS



Challenge: login nodes or compute nodes?

Login nodes

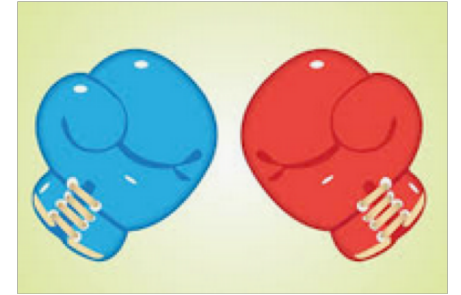
- Pros
 - Available “on demand”
 - Access to filesystems (cf. external VM)
 - Cray programming environment (cf. external VMs)
 - “Free” to the user
- Cons
 - Performance – shared resources!
 - Stability
 - Non trivial to provide parallel contexts

Compute nodes

- Pros
 - Performance - dedicated resources
 - Access to filesystems
 - Cray programming environment
 - Parallel computation (MPI or distributed dask)
 - Production-like execution environment, can T&D with small multi-node notebooks before scaling up
- Cons
 - Difficult to provide “on demand”
 - Not “free”
 - User must remember to close session
 - If allocation ends notebook changes lost

Challenge: Batch vs interactive computing

- How can we reconcile the apparent contradiction between **batch computing** and **interactive computing**?
- Batch is not going away (at least in the immediate term!)
- Reservations?
- Job pre-emption?
- Suspend/Resume?



Jupyter software stack at CSCS

- Installed with EasyBuild
- Based on `cray-python/3.X`
 - provides numpy and scipy that call `cray-libsci` routines
- Pin specific versions of python dependencies
 - assists in maintainability
- Parallel computing available in the notebook
 - ipyparallel (MPI with mpi4py)
 - distributed dask

```
75 lines (59 sloc) | 2.46 KB
Raw Blame History

1  # @author: robinson, sarafael
2
3  easyblock = 'Bundle'
4
5  name = 'jupyterlab'
6  version = '0.35.2'
7
8  py_maj_ver = '3'
9  py_min_ver = '6'
10 py_rev_ver = '5.1'
11
12 pyver = '%s.%s.%s' % (py_maj_ver, py_min_ver, py_rev_ver)
13 pyshortver = '%s.%s' % (py_maj_ver, py_min_ver)
14
15 homepage = 'https://github.com/jupyterlab/jupyterlab'
16 description = "An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architec
17
18 toolchain = {'name': 'CrayGNU', 'version': '18.08'}
19 toolchainopts = {'pic': True, 'verbose': False}
20
21 dependencies = [
22     ('jupyter', '1.0.0'),
23     ('jupyterhub', '0.9.4')
24 ]
25
26 # bundle of Python packages
27 exts_defaultclass = 'PythonPackage'
28
29 exts_list = [
30     ('jupyterlab_server', '0.2.0', {
31         'req_py_majver': '3',
32         'req_py_minver': '6',
33         'use_pip': True,
34         'source_urls': ['https://pypi.python.org/packages/source/j/jupyterlab_server/'],
35     }),
36     (name, version, {
37         'req_py_majver': '3',
38         'req_py_minver': '6',
39         'use_pip': True,
40         'installopts': ' --install-option=--skip-npm ',
41         'source_urls': ['https://pypi.python.org/packages/source/j/jupyterlab/'],
42     }),
43 ]
```



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich


Use cases to demonstrate functionality

Multi-node notebook with MPI – Demonstrator: “Arbor”

- Arbor is a high-performance library for computational neuroscience simulations
 - Developed by colleagues at CSCS, Jülich and BSC as part of the HBP
 - Aim is to prepare neuroscience users for new HPC architectures
 - Arbor is written in C++11 and CUDA (multithreading with TBB, C++11 threads)
 - Python front end; MPI support with mpi4py
- Neuroscientists are not necessarily comfortable with ssh / terminal
- Jupyter thus provides a perfect teaching framework
- Through a simple notebook neuroscientists can
 - describe a neuron model using a **recipe**
 - get resources, create a **parallel execution context**, partition and **load balance**
 - **initiate the simulation** over the distributed system and run the simulation
 - set up **measurement meters**, get **spikes recorded**, show the spiking times of the cells
 - change parameters and immediately see the effect on the results

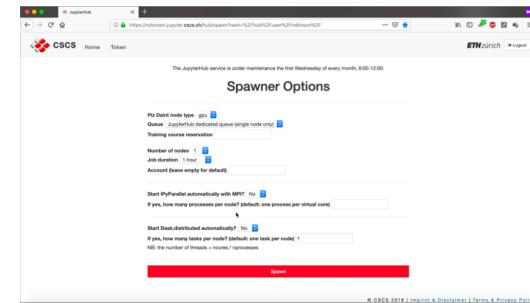
Multi-node notebook with MPI – Demonstrator: “Arbor”

- IPyParallel consists of
 - a controller
 - one or more engines
 - designed to integrate with MPI libraries
- Launch the **ipcontroller** executable on the first compute node of the allocation
- Then launch **ipengines** on all nodes (with srun), providing the IP address of the node running the **ipcontroller**
- Done behind the scenes if user requests MPI in their notebook at spawn time

Start IPyParallel automatically with MPI? ☒ Yes 

If yes, how many processes per node? (default: one process per virtual core)

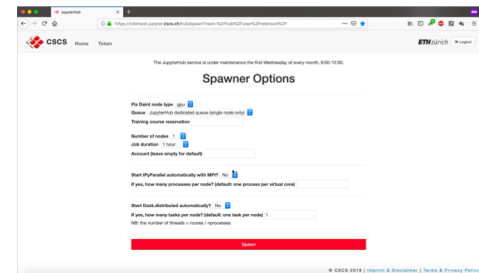
Let's see it in action, interactively (-ish)



MPI scaling up to production – Demonstrator: “Arbor”

- Fully interactive production environment is fine for a single compute node (or a few compute nodes)
- Users want to play in their notebook and – when ready – launch a production batch job, interacting with it through their notebook
- Two options to launch and connect to an external Slurm job
 - **salloc** and then start the **ipcontroller** and **ipengines** by hand
 - NERSC developed **%ipcluster** magic to do this **automagically**
- Examples previously demonstrated (Farrell et al. ISC 2018)
 - Distributed training with MPI via Horovod
 - Hyper-parameter optimization – train and evaluate various models in parallel
- Launch production Arbor simulation using modified **%ipcluster** magic

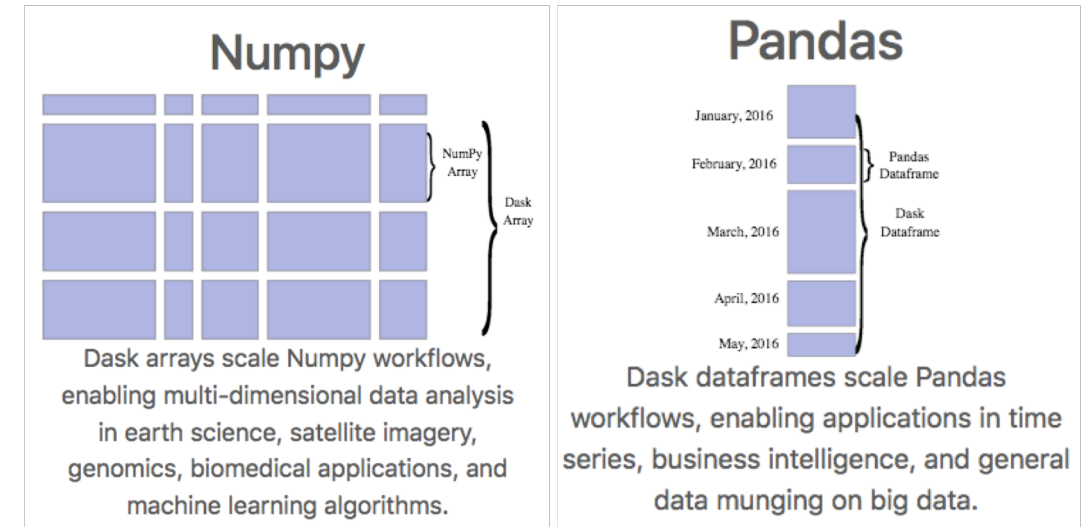
Let's see it in action, interactively (-ish)




Multi-node notebook with Dask Distributed



- **Dask** provides multi-core execution on larger-than-memory data using blocked algorithms and task scheduling: scale python without rewriting code
- Supports the **Pandas** dataframe and **Numpy** array data structures
- A dask distributed network consists of
 - a scheduler node
 - one or more worker nodes
- Launch the **dask-scheduler** executable on the first compute node of the allocation
- Then launch **dask-worker** on all nodes, providing the address (IP, port) to the node that hosts the **dask-scheduler**
- Done behind the scenes if user requests dask distributed at spawn time



Start Dask.distributed automatically? ☒ Yes 

If yes, how many tasks per node? (default: one task per node)

NB: the number of threads = ncores / nprocesses

Spawn



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Providing users with a customizable environment

User customization: virtual environments

- Users can enable virtual environments in JupyterHub by activating them in:

```
${HOME}/.jupyterhub.env
```

- Sourced just before the singleuser-notebook server is launched
- Users can also “`module load <modulefile(s)>`”

- Example: create a virtual environment for TensorFlow

```
> module load daint-gpu jupyterlab
> python3 -m venv --system-site-packages tf
> source ~/tf/bin/activate
(tf)> pip install <required_modules> --user
```

User customization: kernels

- Users can install their own kernels in `${HOME}/.local/share/jupyter/kernels`

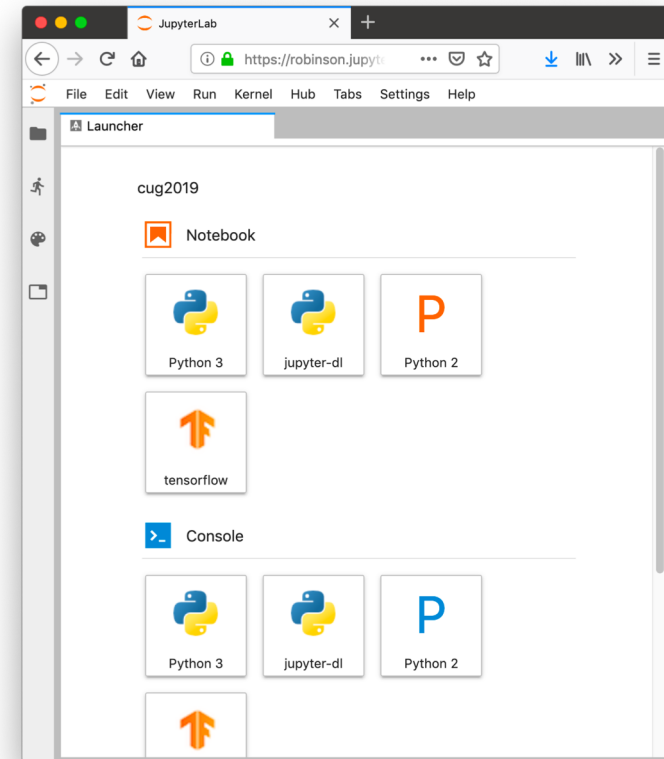
```
(tf)> export PYTHONPATH=~/.tf/lib/python3.6/site-packages:${PYTHONPATH}
(tf)> pip install --ignore-installed ipykernel --user
(tf)> python3 -m ipykernel install --user --name=tensorflow
```
- Write a launch script `~/tf/kernel.sh` for the kernel to activate the venv, load modules, etc.

```
#!/bin/bash
module load TensorFlow/1.12.0-CrayGNU-18.08-cuda-9.1-python3
module load Horovod/0.16.0-CrayGNU-18.08-tf-1.12.0
source ~/.tf/bin/activate
export PYTHONPATH=~/.tf/lib/python3.6/site-packages:${PYTHONPATH}
exec python -m ipykernel $@
```

User customization: kernels

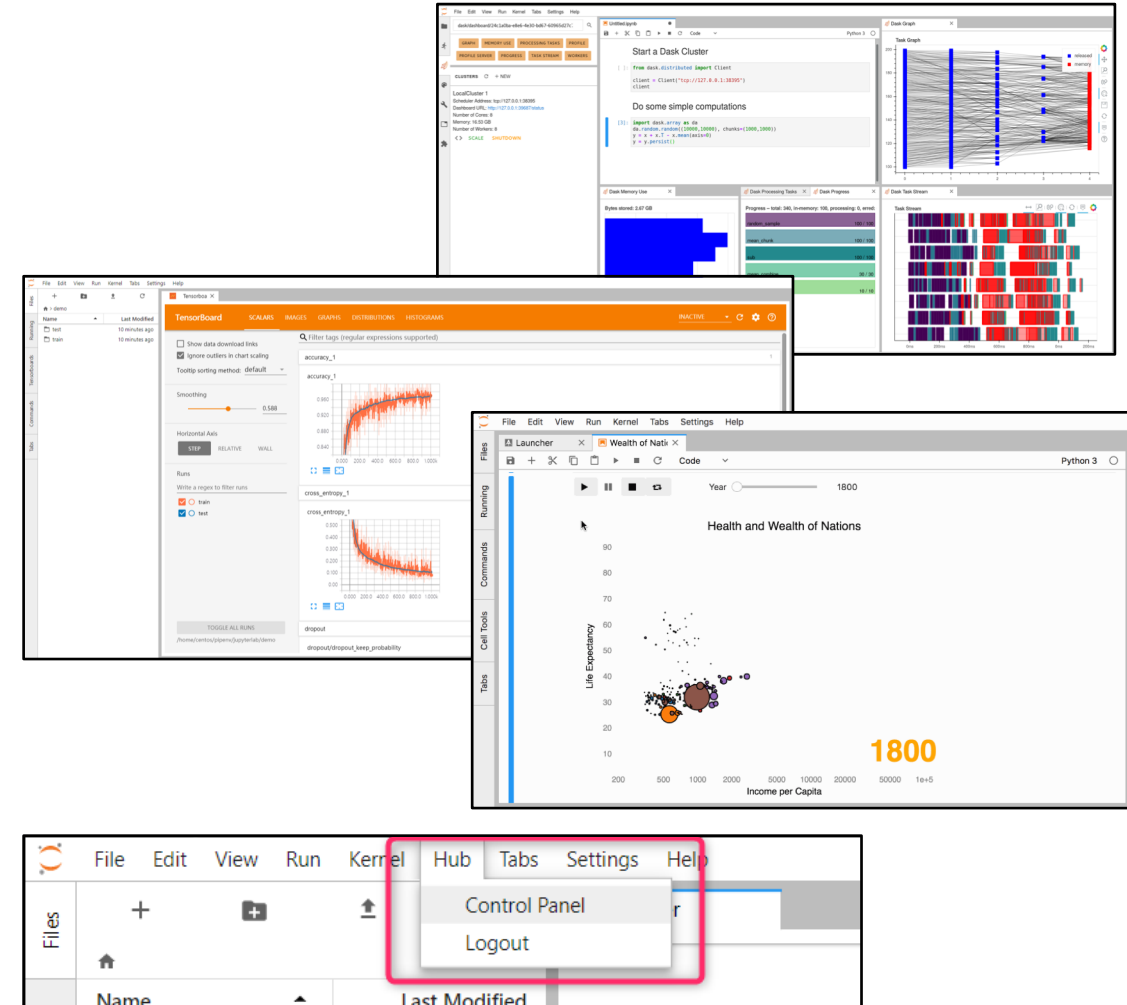
- Finally, edit the kernel file:

```
> cat ~/.local/share/jupyter/kernels/tensorflow/kernel.json
"argv": [
  "~/tf/kernel.sh",
  "-f",
  "{connection_file}"
],
```



User customization: JupyterLab Extensions

- Fundamentally, JupyterLab is designed as a customizable, extensible environment
- Extensions can provide new themes, file viewers and editors, and renderers for rich output
- 100 GitHub repos tagged “jupyterlab-extensions”
 - `dask-labextension`
 - `jupyterlab-tensorboard`
 - `bqplot`
 - `jupyterlab-hub` – adds a Hub menu to JupyterLab to allow users to log out of JupyterHub or access the control panel



User customization: JupyterLab Extensions

- The “classic” notebook allowed users to install extensions (`--user`) and has a hierarchical prioritization of directories
- But JupyterLab is a **single bundle** using WebPack – if a user has custom extensions they need their own JupyterLab installation!

```
>JUPYTERLAB_DIR=$HOME/<path> jupyter labextension install <extension>
```

- Centrally installed extensions are lost!
- We can't easily provide a centralized installation *and* allow users to add their own extensions on top
- **Challenge:** how can we support such an extensible environment?

Checklist for current implementations

- Development environment? 👍
- Access to data? 👍
- Parallel computing?
 - MPI 👍
 - Distributed dask 👍
- Flexibility and customization 👍 👎
- Supercomputing on demand? 👎

Future plans

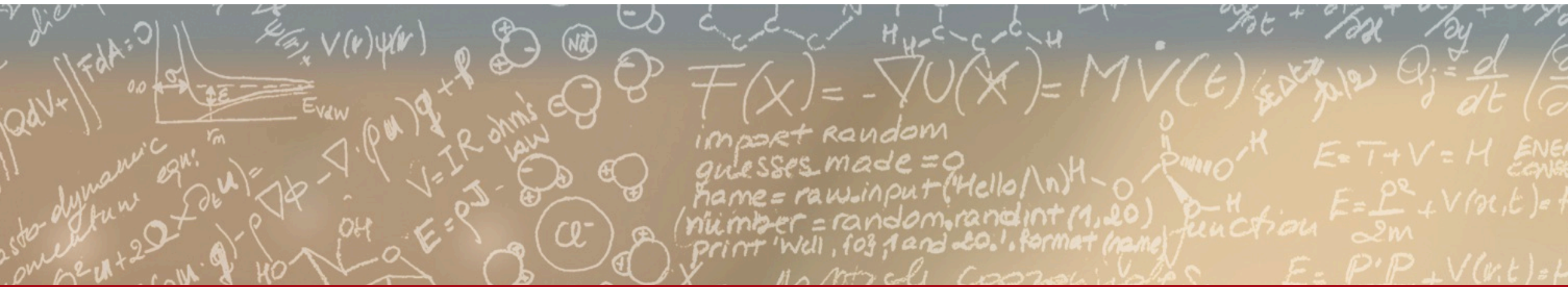
- Notebooks will be spawned on a variety of platforms
 - Piz Daint
 - Cloud infrastructure
 - Future systems
- CSCS hosted cloud infrastructure
 - Early testbed – OpenStack deployment
 - Submit jobs from the Notebook to Piz Daint (via Slurm magics, ipcluster magics, ...)
 - Future scheduling platforms
 - Fixed Kubernetes domain / HPC scheduler domains
 - Elasticity / metascheduling
- Shasta....



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

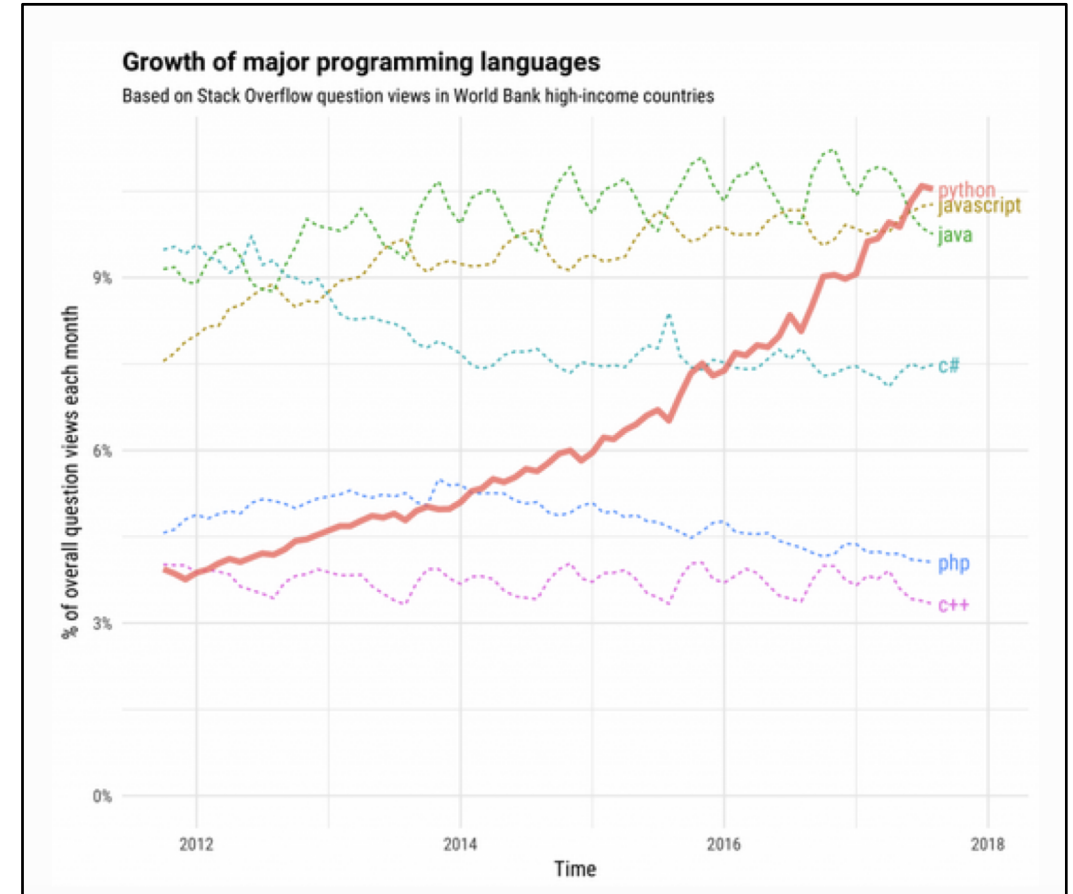


Reserve...

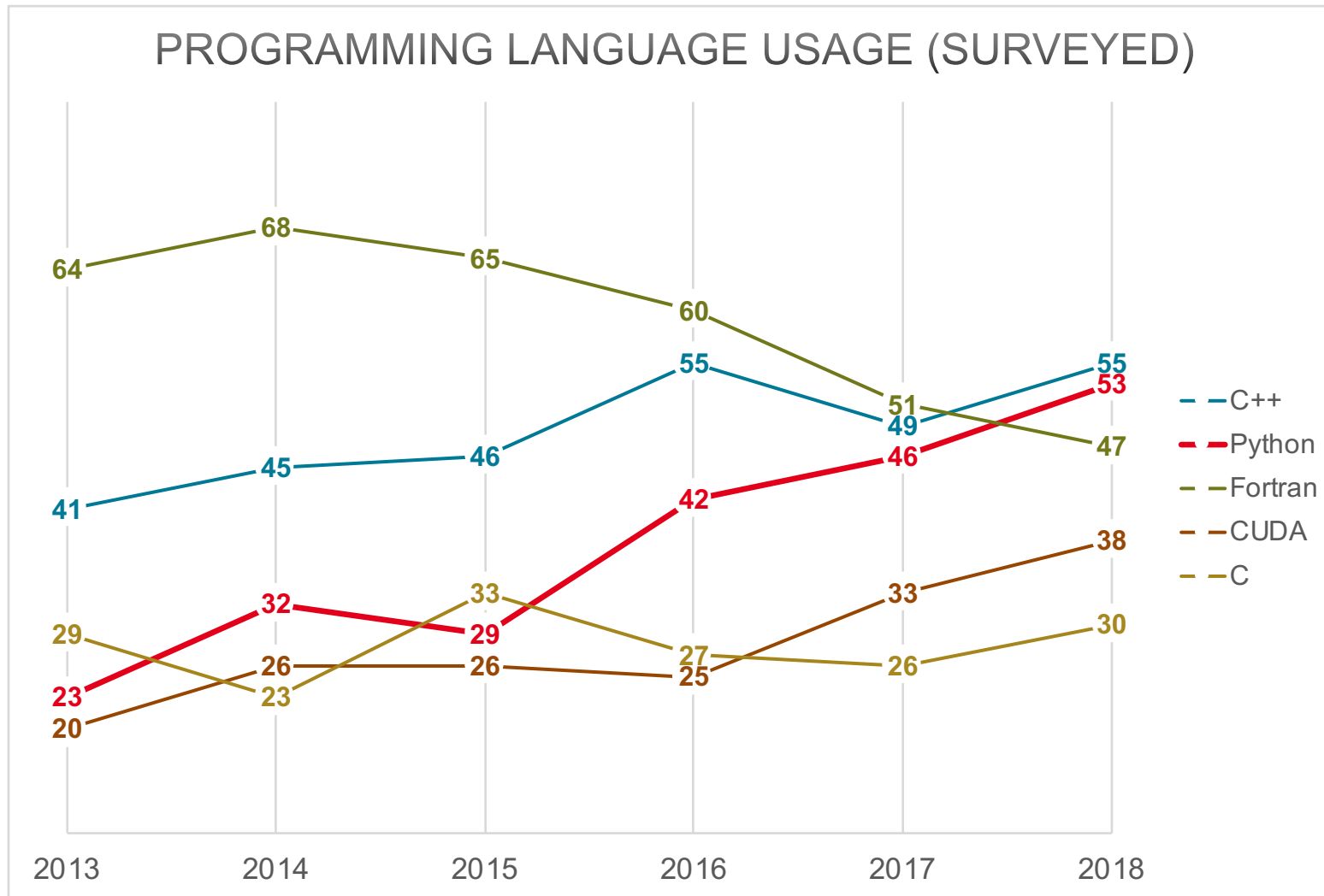
The rise of Python



- Python has grown to become the dominant language both in data analytics and general programming
- Rise fueled by computational libraries like Numpy, Pandas, and Scikit-Learn and libraries for visualization, interactive notebooks, collaboration, etc
- Python long used as glue, for pre- and/or post-processing.. but increasingly used for simulation as well



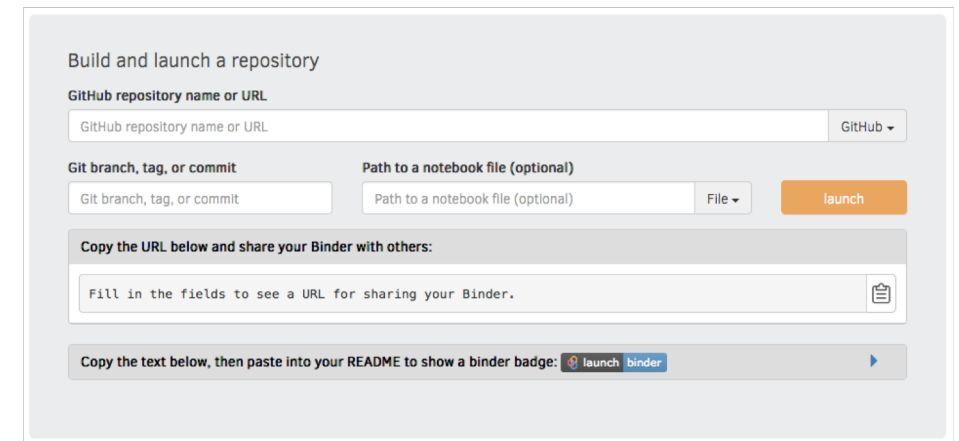
Python usage at CSCS



Us and Them (e.g. Binder)



- Binder provides an executable environment for notebooks in Git repos
- “[makes] your code immediately reproducible by anyone, anywhere.”
- Click a URL, interact with someone else’s code, execute it directly in the cloud
- Reference deployment of BinderHub at mybinder.org, and it’s **free!**
- How does it work?
 - Creates containers from repos (repo2docker)
 - Creates user sessions to serve them (JHub)
 - Provides interface to use/share them (BinderHub)
 - Provides a free public service (mybinder.org)
- losc.ligo.org/tutorials Live! [or recording]

A screenshot of the Binder web interface. It shows a form titled "Build and launch a repository". The form has two main input fields: "GitHub repository name or URL" and "Git branch, tag, or commit". There is also a "Path to a notebook file (optional)" field and a "File" dropdown menu. A "launch" button is visible. Below the form, there is a section for sharing the URL, with a text box that says "Fill in the fields to see a URL for sharing your Binder." and a "Copy" icon. At the bottom, there is a section for a binder badge, with a text box that says "Copy the text below, then paste into your README to show a binder badge:" and a "launch binder" button.

It's awesome right?!



- But hold on...

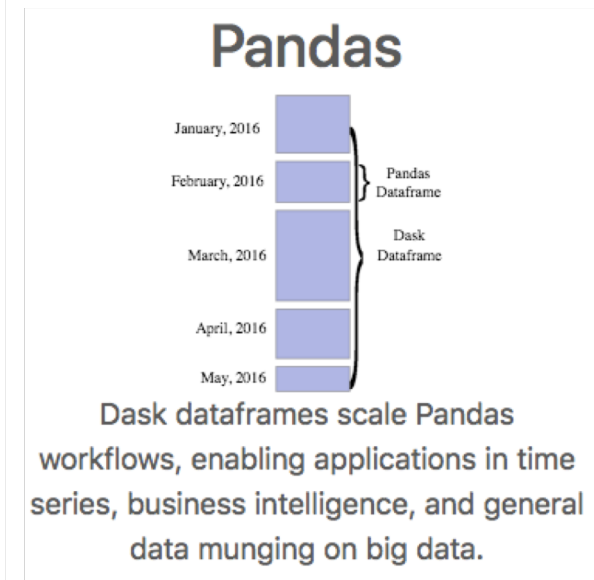
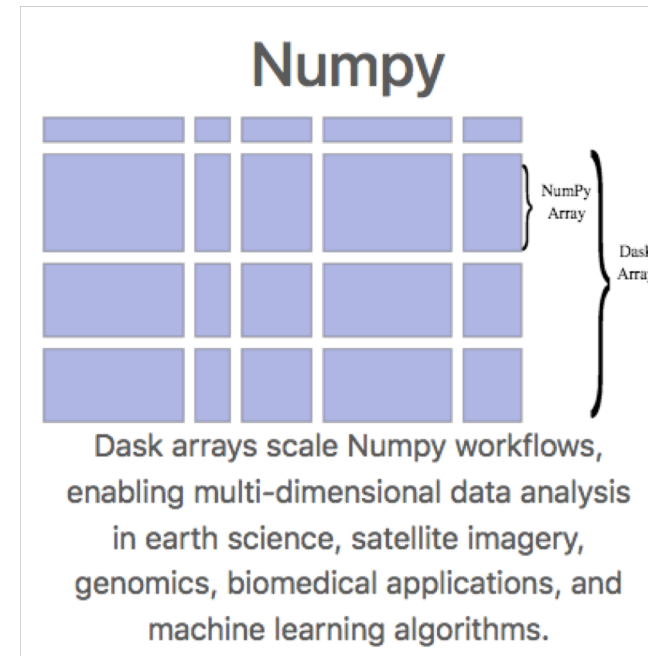
“Binder is a research pilot, whose main goal is to understand usage patterns and workloads for future evolution and development. It is not a service that can be relied on for critical operations.”

- And it's not *really* free...
- And what about my data? – how do I connect to huge data (and fast data!)
- And what about computation? We are HPC right?
- And what about authentication/authorization?
- As HPC centres we'd like to provide something that gives a user experience like Binder, but... we have the batch system, filesystems, security policies, parallel computing...

Multi-node notebook with Dask Distributed – Demonstrator: “Enron”




- **Dask** provides multi-core execution on larger-than-memory data using blocked algorithms and task scheduling: scale python without rewriting code
- Supports the **Pandas** dataframe and **Numpy** array data structures
- Dask can run on a local computer or be scaled up to a cluster
- Key dask component: **Dask Bag**
 - Dask bag is able to store and process collections of Pythonic objects that are unable to fit into memory. Dask Bags are great for processing logs and collections of json documents
- **Enron Corpus** is a dataset of 600K emails related to the investigation of the collapse
- One of the few publicly available mass collections of real email



Multi-node notebook with Dask Distributed – Demonstrator: “Enron”

- A dask distributed network consists of
 - a scheduler node
 - one or more worker nodes
- Launch the **dask-scheduler** executable on the first compute node of the allocation
- Then launch **dask-worker** on all nodes, providing the address (IP, port) to the node that hosts the **dask-scheduler**
- Done behind the scenes if user requests dask distributed at spawn time

Start Dask.distributed automatically? 

If yes, how many tasks per node? (default: one task per node)

NB: the number of threads = ncores / nprocesses

Spawn

Let's see it in action, interactively (-ish)

