Python for HPC

Andrea Zonca - SDSC

Jupyter Notebook

Data exploration in your browser

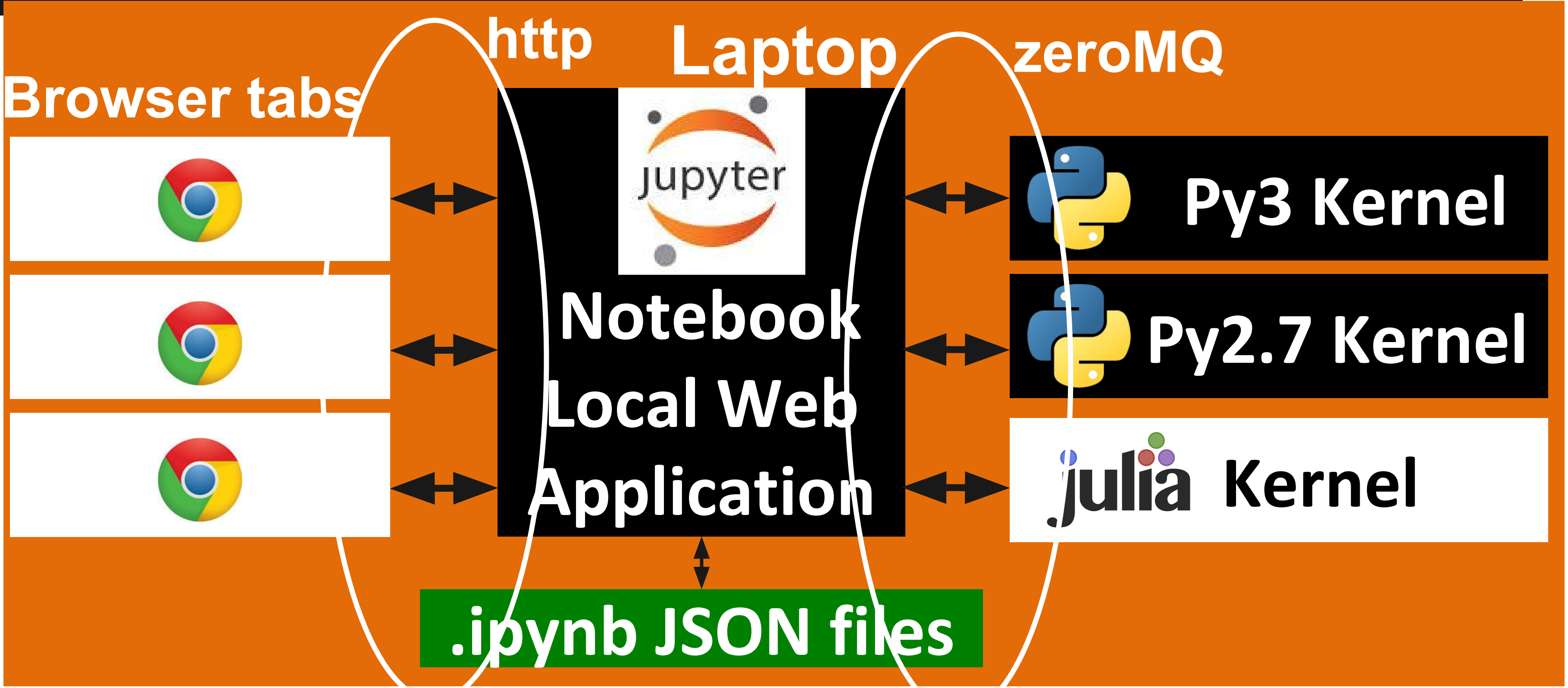
What is the notebook?

- Browser based interactive console
- Supports multiple sessions in browser tabs
- Each session has a Kernel executing computation
- Saved in JSON format

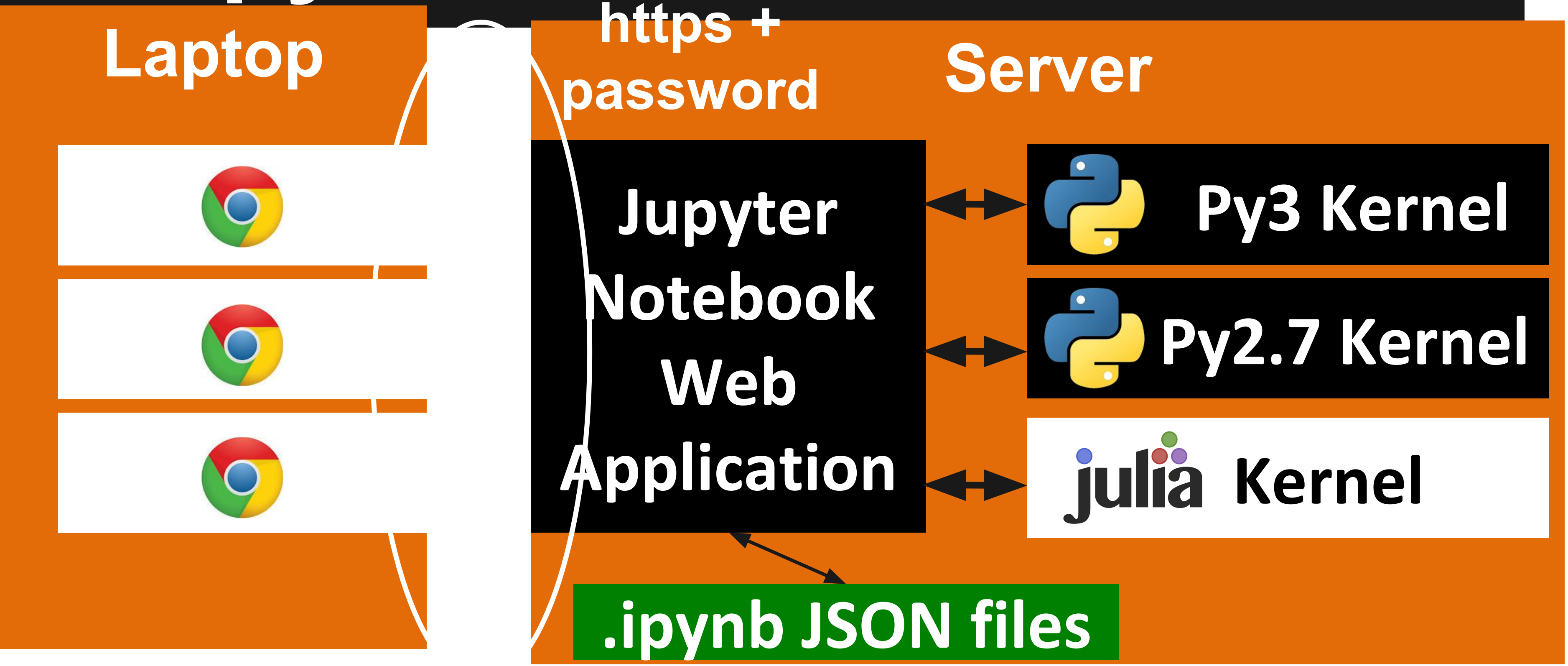
Notebooks on Nature

http://www.nature.com/news/interactive-notebooks-sharing-the-code-1.16261

Jupyter notebook local



Jupyter notebook remote



Laptop

https + password

Server

Jupyter Notebook Web Application

Py3 Kernel

Py2.7 Kernel

julia Kernel

.ipynb JSON files

Clone workshop repository

ssh into comet with training account

```
git clone URL workshop
```

URL is https://github.com/sdsc-scicomp/2015-11-12-ucla

Modules on Comet

```
module load python scipy
```

- add line to .bashrc

Setup on Comet

- ssh to Comet

```
salloc --nodes=1 --tasks-per-node=1 -t 02:00:00 --res=ucla2015
```

- `ssh comet-xx-xx`
- `ipython notebook --no-browser --ip="*" &`
- later better setup config file

connect with browser

Open browser on your laptop and connect to comet-xx-xx.sdsc.edu:8888

New -> Notebook

!hostname

More secure setup

http://zonca.github.io/2015/09/ipython-jupyter-notebook-sdsc-comet.html

IPython notebook demo

- Python code
- Formatted text
- Equations
- Plots
- Cells execution, cells order
- Clear output

Why the notebook?

- Literate programming: code and explanation together
- Reproducible science: document easily every step
- Easy to share computations: send one single notebook instead of scripts/plots/.doc

ipynb documents

- JSON format
- includes plots in binary format
- easy to convert to .html/.pdf for sharing
- http://nbviewer.ipython.org
- Recently rendered automatically on Github

HPC: interactive notebooks

- Analyze large amount of data
- In-situ visualization
- Centralized Python stack
- Check long-running computations
- Prepare and submit batch jobs

Notebooks as scripts

- Install runipy:

  ```
  pip install --user runipy
  ```

- Setup .bashrc:

  ```
  cd ~/workshop/python_hpc
  cat setup_pip_local.sh >> ~/.bashrc
  ```

- Restart bash with: `bash`

Notebooks as scripts

- demo of runipy
  - open and execute fit_line.ipynb
  - uncomment cell with (os.environ)
  - white_noise_scale=1000 runipy fit_line.ipynb fit_line_1000.ipynb
  - open fit_line_1000.ipynb, what happened?
- demo of batch submission of SLURM serial runipy jobs using pipes

Hands-on

- Open the notebook interactively
- Add saving the plot with plt.savefig ("figurename.png") in the same cell
- Test with runipy on the interactive node
- Rerun the jobs through the queue

IPython parallel

Parallel computing the easy way

IPython parallel

- High-level API for distributed computing with Python
- Engines (Python worker processes) connected to Controller with ZeroMQ
- Client, i.e. user's IPython session, connects to the Controller

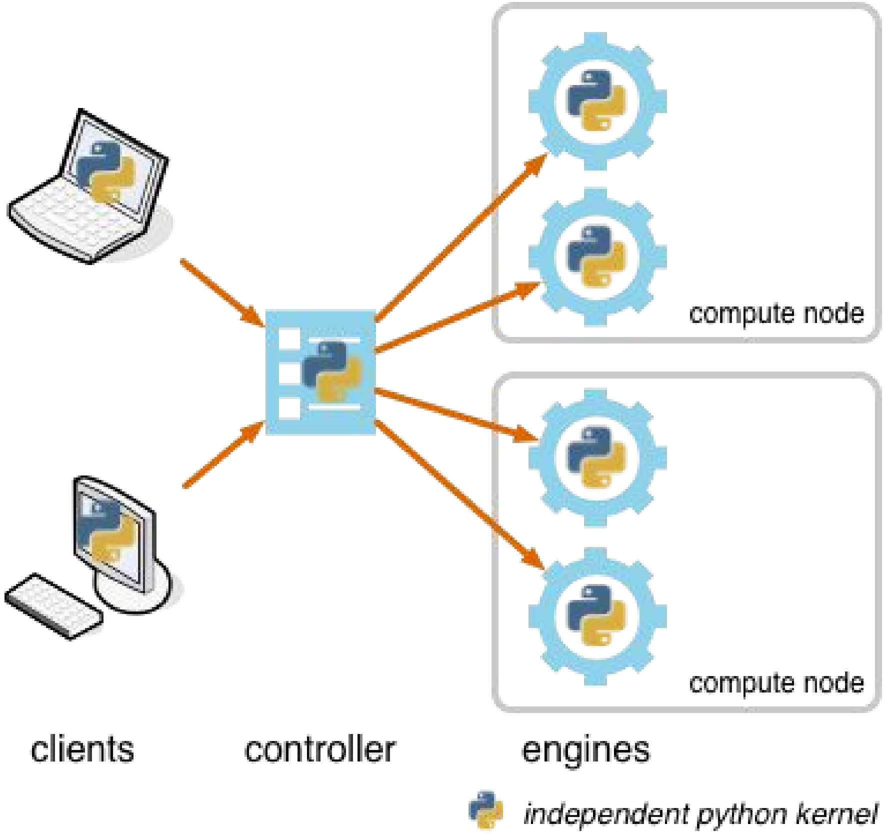


Image by Continuum Analytics

IPython parallel architecture

Functionalities

- Load balanced queue for trivially parallel jobs
- Supports job dependencies
- Direct interface to Engines
- Supports MPI applications, Python or C/C++/Fortran

IPython parallel config

```
ipython profile create
```

```
cp ipython_parallel_configuration/* ~/.ipython/profile_default
```

IPython parallel Demo

- Launch cluster with 48 engines:
  - ipcluster start --n=48
- Connect with IPython Notebook
- Print ids, hostnames
- Launch demo job and check it runs correctly

Hands-on

- Create a duplicate of fit_line.ipynb
- Reformat fit_line code into a single function
- Send it to engines for execution within the balanced queue
- Print out the results from the notebook

IPython parallel and MPI

```python
from mpi4py import MPI
import numpy as np

def psum(a):
    locsum = np.sum(a)
    rcvBuf = np.array(0.0,'d')
    MPI.COMM_WORLD.Allreduce([locsum, MPI.DOUBLE],
        [rcvBuf, MPI.DOUBLE],
        op=MPI.SUM)
    return rcvBuf
```

```
In [1]: from IPython.parallel import Client

In [2]: c = Client()

In [3]: view = c[:]

In [4]: view.activate() # enable magics

# run the contents of the file on each engine:
In [5]: view.run('psum.py')

In [6]: view.scatter('a',np.arange(16,dtype='float'))

In [7]: view['a']
Out[7]: [array([ 0.,  1.,  2.,  3.]),
         array([ 4.,  5.,  6.,  7.]),
         array([  8.,   9.,  10.,  11.]),
         array([ 12.,  13.,  14.,  15.])]

In [7]: %px totalsum = psum(a)
Parallel execution on engines: [0,1,2,3]

In [8]: view['totalsum']
Out[8]: [120.0, 120.0, 120.0, 120.0]
```