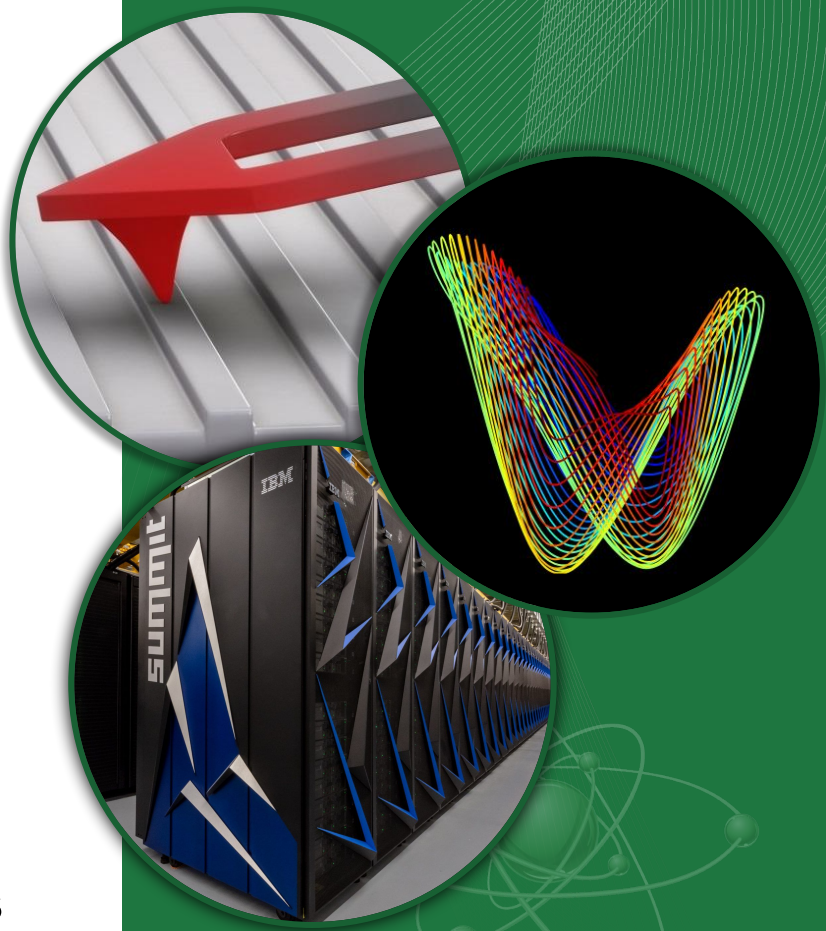
The Jupyter Ecosystem & its Impact on Data-Analytics & Science



03/07/2019

Suhas Somnath

Advanced Data and Workflows Group
National Center for Computational Sciences

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

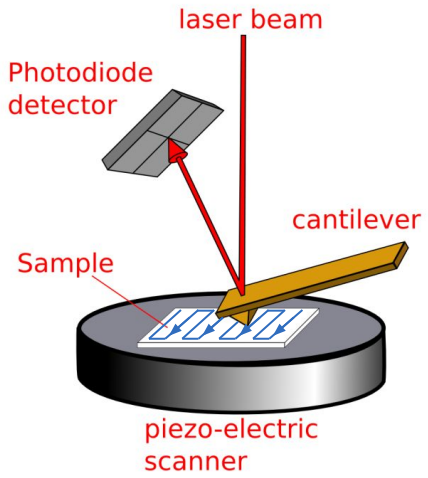
ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

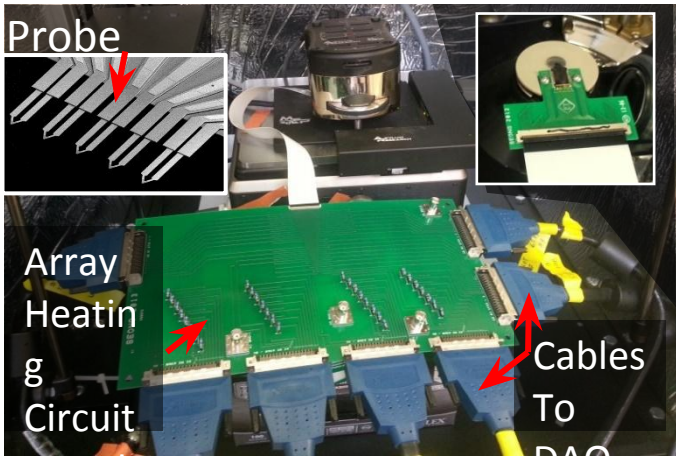
About me…

- PhD in Mechanical Engineering (2014)
- Postdoctoral Researcher (2014 - 2017)
- Computer / Data Scientist (2017 – Present)

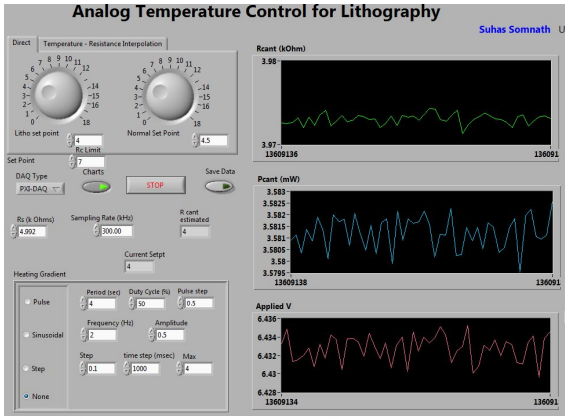
Atomic Force Microscopy



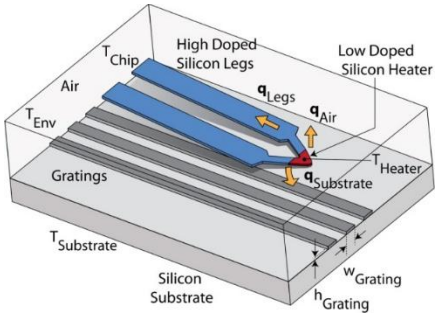
Instrument Development



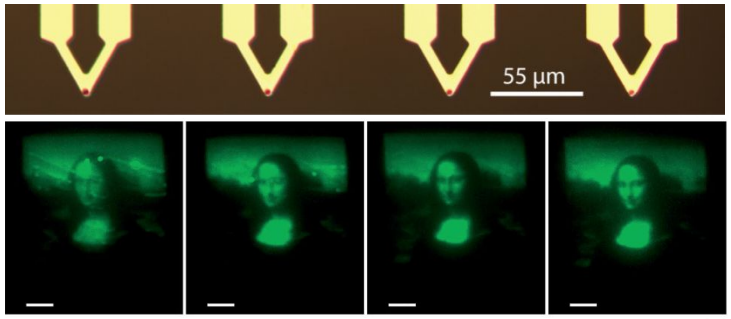
Instrumentation Software



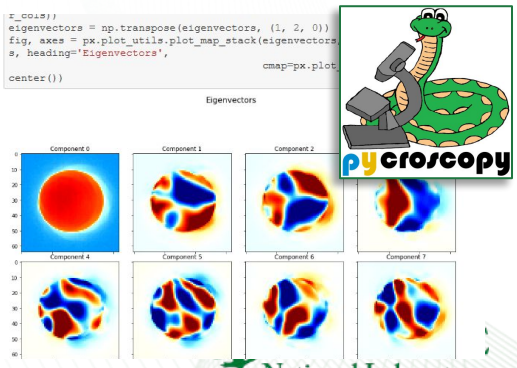
Simulation / Modeling



New measurement & manufacturing techniques

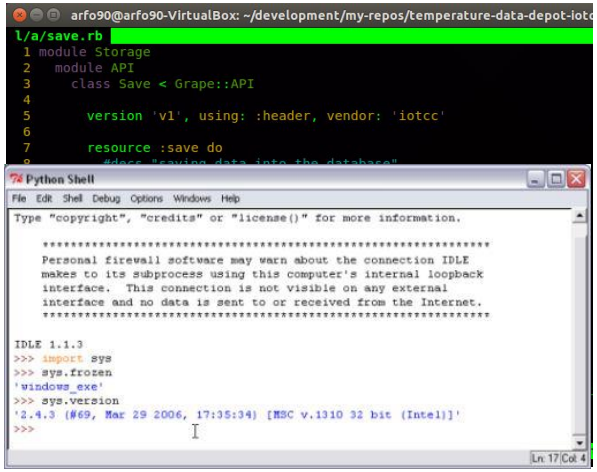


Data Analysis Software



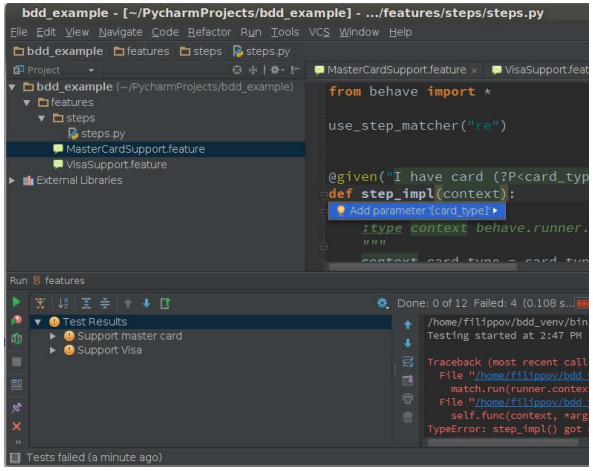
Prototyping, Disseminating, Exploring in Python

Editor + Python Console



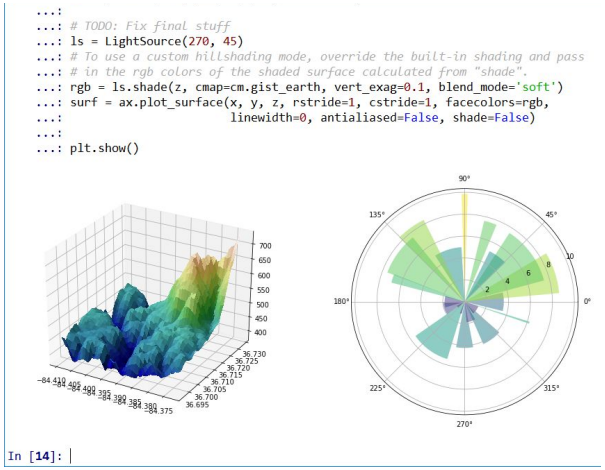
- Fast
- Cumbersome – multiple windows (code, results)
- No markdown, equations
- Hard to follow / teach
- Minimal high-level tools - auto-complete

IDE



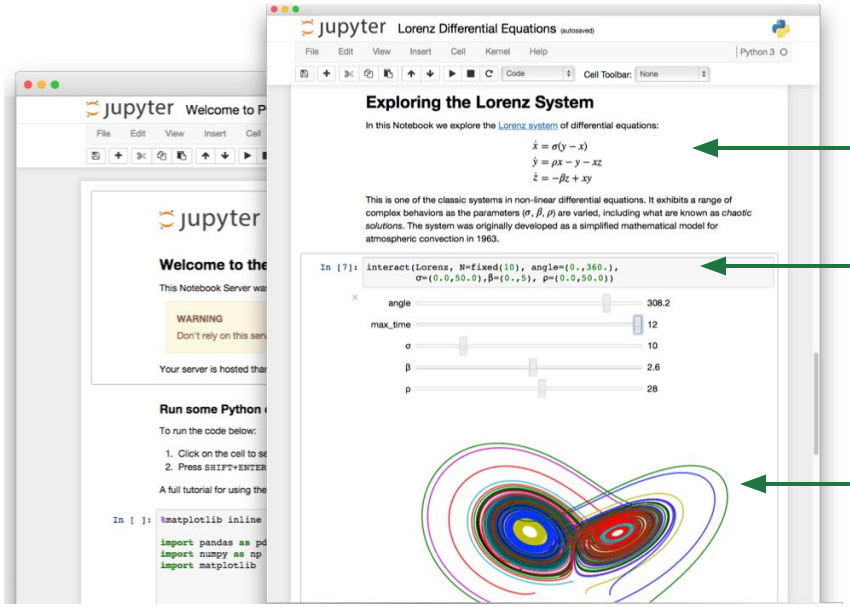
- Very powerful
- Overkill for prototyping / teaching / learning
- No markdown, equations …
- Cumbersome – multiple windows*
- Hard to follow

iPython console



- Light and simple
- Helpful tools like autocomplete, etc.
- In-line plots and results!
- Single document
- No markdown, equations …
- Hard to export

Jupyter Notebooks



- Interactive documents
- Exploratory programming
- Markdown text
  - Equations
- Code
- Images
- Interactive – slice through data, pan, move, rotate …

Jupyter Notebooks

<LIVE DEMO>

Notebooks are great for

- Exploratory data analysis
  - Try different parameters / algorithms quickly
  - Keep notes
- Code prototyping
  - Develop snippets of code that can become part of a package
- Computational narrative
  - Leverage markdown capabilities to explain what, why, how, etc.
  - Cause (code + data) and effect (plots, output)

Notebooks for Teaching

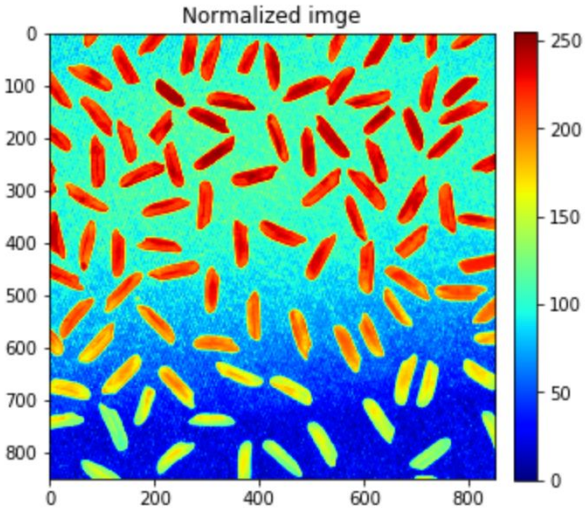
github.com

Thresholding ¶

In certain cases, we may may be working on images that have two distinct phases, kinds of objects, heights, etc. **that appear different to a reasonable extent** (like the rice image) and we may want to apply separate processing steps on each part of the image.

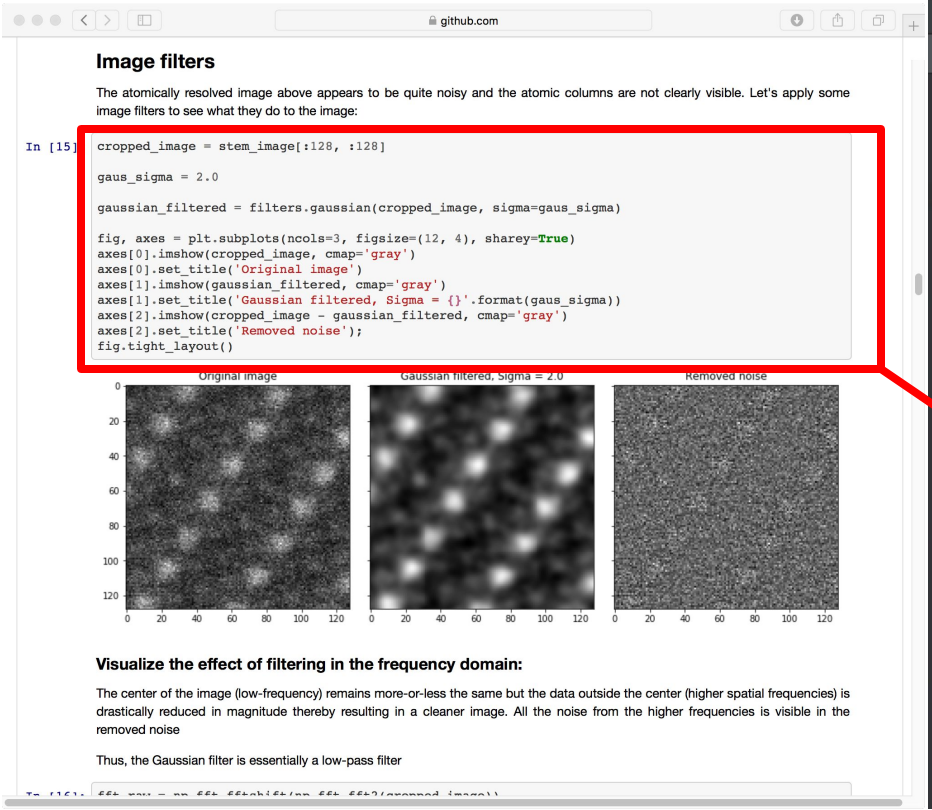
In [25]:
```
# Normalizing to [0, 255]
rice_renormalized = rice_image - rice_image.min()
rice_renormalized = 255 * rice_renormalized / rice_renormalized.max()

fig, axis = plt.subplots(figsize=(5, 5))
im_handle = axis.imshow(rice_renormalized, cmap='jet')
# Add a colorbar
cbar = plt.colorbar(im_handle, ax=axis, fraction=0.046, pad=0.04)
axis.set_title('Normalized imge');
```

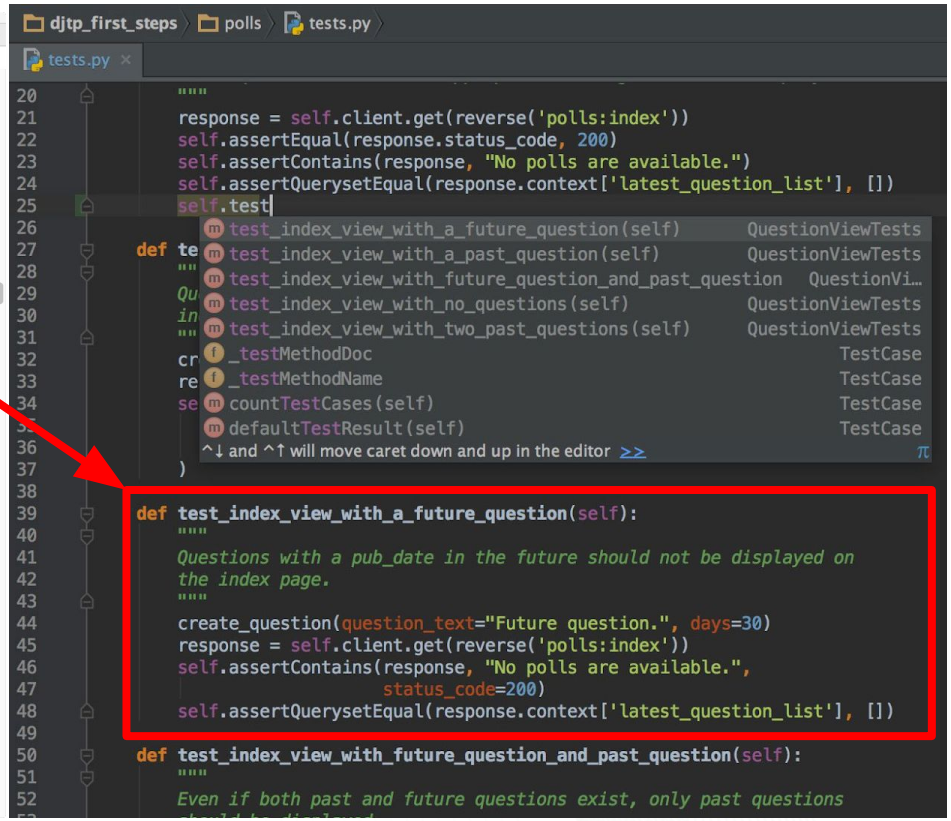


Notebooks for Exploration / Prototyping

Exploratory data analysis



Prototyping code



- Try parameters / algorithms quickly
- Keep notes

Develop snippets of code that can become part of a package

Towards Open & Reproducible Analytics / Science

Academic papers accompanied with:

- Jupyter notebooks that show all analysis (raw data → figures)
- Data with DOI number
- Software environment → containers

Jupyter notebook associated with paper

Visualizing Filtering Results

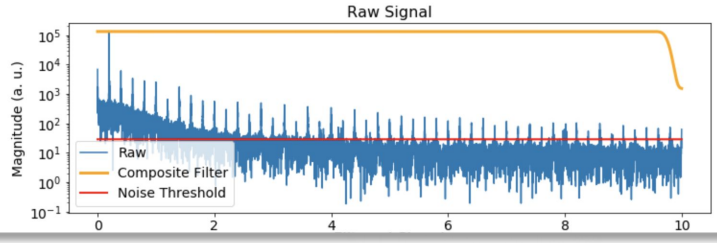
Run the next cell to see what the IV curves look like for the chosen filter parameters. If the current set of filter parameters do not work as well, change the parameters in the previous cell and try again.

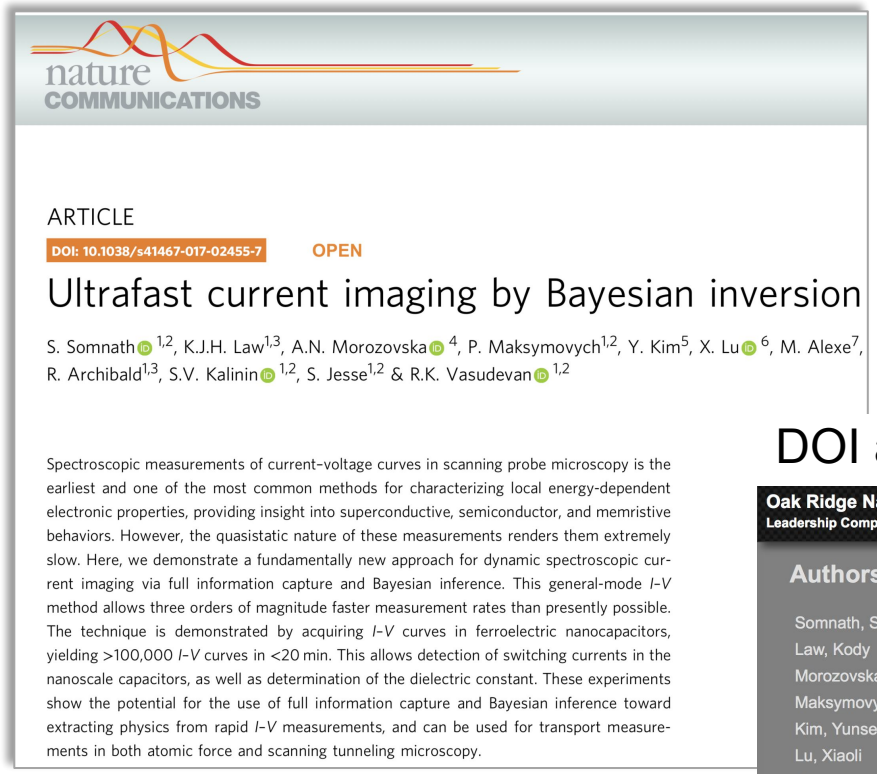
Once the results look good, proceed to filter the entire dataset

In [7]:
```
# Test filter on a single line:
row_ind = 50
filt_line, fig_filt, axes_filt = px.processing.gmode_utils.test_filter(h5_main[row_ind], filte
r_parms, samp_rate,
                                                                       show_plots=True, use_rai
nbow_plots=False)
fig_filt.savefig(os.path.join(other_figures_folder, 'FFT_filter_on_line_{}.png'.format(row_ind
)), format='png', dpi=300)

raw_row = np.reshape(h5_main[row_ind], (-1, pts_per_cycle))
filt_row = filt_line.reshape(-1, pts_per_cycle)

fig, axes = px.plot_utils.plot_loops(single_AO, [raw_row, filt_row], dataset_names=['Raw', 'Fi
ltered'],
                                     line_colors=['r', 'b'], x_label='Bias (V)', title='FFT Fi
ltering',
                                     plots_on_side=3, y_label='Current (nA)',
                                     subtitles='Row: ' + str(row_ind) + ' Col:')
fig.savefig(os.path.join(other_figures_folder, 'Example_filtered_loops_from_line_{}.png'.forma
t(row_ind)), format='png', dpi=300)
```





nature COMMUNICATIONS

ARTICLE

DOI: 10.1038/s41467-017-02455-7 OPEN

Ultrafast current imaging by Bayesian inversion

S. Somnath[1,2], K.J.H. Law[1,3], A.N. Morozovska[4], P. Maksymovych[1,2], Y. Kim[5], X. Lu[6], M. Alexe[7], R. Archibald[1,3], S.V. Kalinin[1,2], S. Jesse[1,2] & R.K. Vasudevan[1,2]

Spectroscopic measurements of current–voltage curves in scanning probe microscopy is the earliest and one of the most common methods for characterizing local energy-dependent electronic properties, providing insight into superconductive, semiconductor, and memristive behaviors. However, the quasistatic nature of these measurements renders them extremely slow. Here, we demonstrate a fundamentally new approach for dynamic spectroscopic current imaging via full information capture and Bayesian inference. This general-mode *I*–*V* method allows three orders of magnitude faster measurement rates than presently possible. The technique is demonstrated by acquiring *I*–*V* curves in ferroelectric nanocapacitors, yielding >100,000 *I*–*V* curves in <20 min. This allows detection of switching currents in the nanoscale capacitors, as well as determination of the dielectric constant. These experiments show the potential for the use of full information capture and Bayesian inference toward extracting physics from rapid *I*–*V* measurements, and can be used for transport measurements in both atomic force and scanning tunneling microscopy.

DOI associated with data (raw → paper figures)

Oak Ridge National Laboratory
Leadership Computing Facility

10.13139/OLCF/1410993

Download

Authors

Somnath, Suhas	somnaths@ornl.gov
Law, Kody	lawkj@ornl.gov
Morozovska, Anna	anna.n.morozovska@gmail.com
Maksymovych, Petro	maksymovychp@ornl.gov
Kim, Yunseok	ykim943@gmail.com
Lu, Xiaoli	xllu@xidian.edu.cn
Alexe, Marin	M.Alexe@warwick.ac.uk

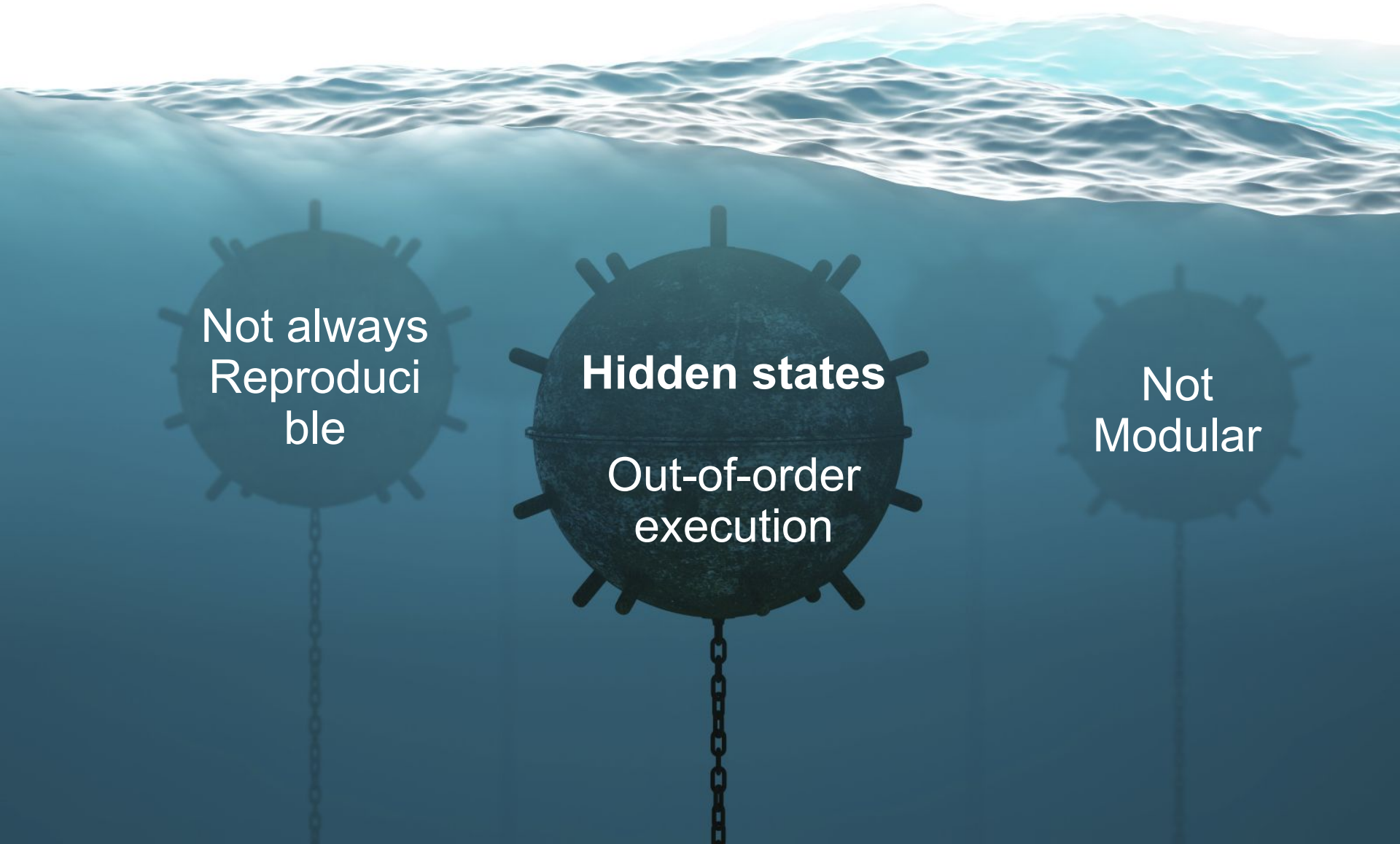
OAK RIDGE National Laboratory

Notebooks for Narratives

- Example notebooks that supplement academic articles
  - https://pycroscopy.github.io/pycroscopy/papers_conferences.html#journal-papers-using-pycroscopy

Is it all smooth sailing with Notebooks?

Plenty of Caveats with Notebooks



Notebooks encourage

Hidden State

jupyter madness

File Edit View Insert Cell Kernel Help Python 3

Code CellToolbar

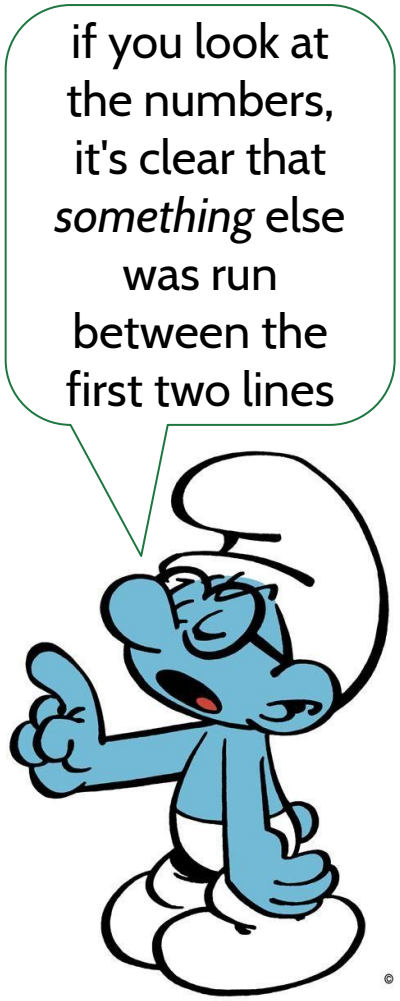
```
In [1]: def f(x): return x + 2

In [3]: y = f(2)

In [4]: y == 4
Out[4]: False

In [5]: print(y)
        5
```

- Deleted / updated cells
- Out-of-order execution of code
- Confusing for newcomers



@joelgrus #jupytercon

Questionable Reproducibility

In [1]:

```
import torch
import torch.nn as nn
from torch.autograd import Variable

import pandas as pd
import random
import string
import numpy as np

import sys, os

import torch.utils.data as data

os.environ["CUDA_VISIBLE_DEVICES"] = '0'

all_characters = string.printable
number_of_characters = len(all_characters)

artists = [
'ABBA',
'Ace Of Base',
'Aerosmith',
'Avril Lavigne',
'Backstreet Boys',
'Bob Marley',
'Bon Jovi',
'Britney Spears',
'Bruno Mars',
'Coldplay',
'Def Leppard',
'Depeche Mode',
'Ed Sheeran',
```

pytorch 0.3? 0.4? 0.4.1?

I suppose I could guess based on the commit date

Will it work on my machine?

- Hardware
- Operating system
- Python version?

@joelgrus #jupytercon

Questionable Reusability

In [1]:
```
import torch
import torch.nn as nn
from torch.autograd import Variable

import pandas as pd
import random
import string
import numpy as np

import sys, os

import torch.utils.data as data

os.environ["CUDA_VISIBLE_DEVICES"] = '0'

all_characters = string.printable
number_of_characters = len(all_characters)

artists = [
'ABBA',
'Ace Of Base',
'Aerosmith',
'Avril Lavigne',
'Backstreet Boys',
'Bob Marley',
'Bon Jovi',
'Britney Spears',
'Bruno Mars',
'Coldplay',
'Def Leppard',
'Depeche Mode',
'Ed Sheeran',
```

Why are you doing things this way?
Instructions?

Can I use my data instead?

What if I only want to use some portions?
Can't import (easily)

Typical excuse...

[redacted]

@[redacted]

Follow

Data science code doesn't need to follow the rules of good software engineering, because data science is not about creating software but about experimenting with building prototypes of models. 👉 Great tip from @jeremyphoward

4:18 AM - 2 May 2018

4 Retweets **26** Likes

@joelgrus
#jupytercon

Other challenges

- Importing notebooks in other notebooks quickly becomes ugly
- Same with notebooks sharing state with others
- Git diffs are filled with JSON changes / cruft

Notebooks are not good for

- Teaching / following code
  - Out-of-order execution of code
  - Deleted / updated cells
- Software development
  - Not modular / reusable
  - Challenging / tedious to unit test
  - Importing notebooks quickly becomes ugly
  - Git often catches JSON changes rather than code / results
  - Collaboration is tedious
- Reproducibility
  - Dependencies not captured

Not the end of the world for Jupyter

- Follow best practices
- Use notebooks as they were intended

Ten Simple Rules for Reproducible Research in Jupyter Notebooks

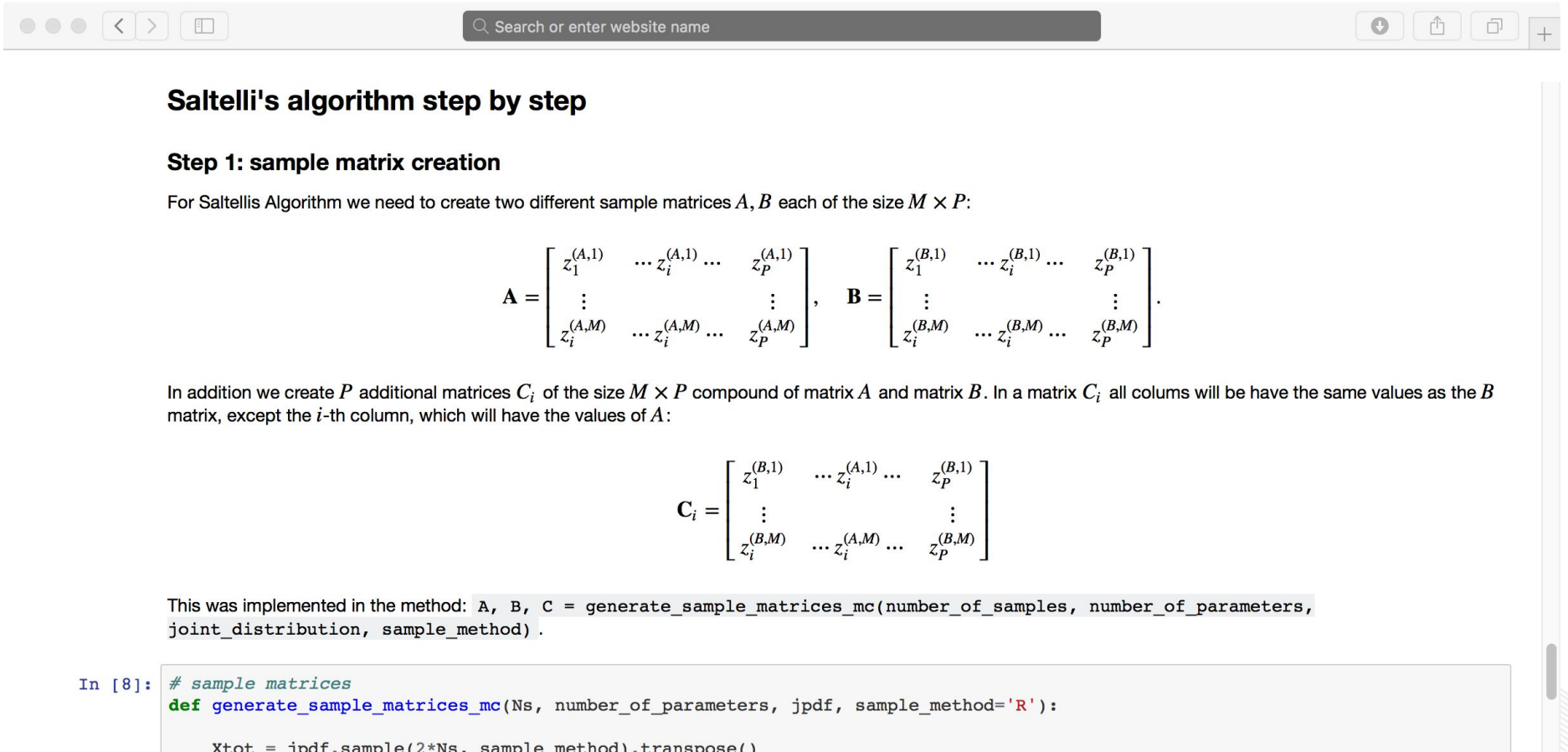
Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H. Nguyen, Sara Brin Rosenthal, Fernando Pérez, Peter W. Rose

arXiv preprint arXiv:1810.08055. 2018 Oct 13.

https://github.com/jupyter-guide/ten-rules-jupyter

1. Tell a Story for an Audience

- Leverage markdown to tell a compelling story
- Why are you doing things this way?
- What do the results mean and why?



2. Document the Process, not just Results

- Document before you forget!
- Name, contact, date(s) …
- Both successful and failed experiments
- Where you got a block of code from
- Avoid manipulating figures etc. using external software

3. Add Divisions to Make Steps Clear

- Avoid one-line and very long cells
- Each cell -> one meaningful step
- Label cells using markdown above
- Split long notebooks into multiple notebooks

4. Modularize Code

- Avoid multiple copies of code with alterations
  - Wrap into function
  - Turn into module / package

5. Record Dependencies

- Explicitly print out dependencies using extensions
  - Watermark (https://github.com/rasbt/watermark)

- Write out `requirements.txt`
  - Easier to deploy containers. E.g. – Binder (more later)

6. Use Version Control

- Track modifications, source of bugs, etc.

- Notebook-specific diffing tool, e.g. – `nbdime`
  - Understands notebook structure
  - Presents meaningful diffs – minimal JSON metadata
  - https://github.com/jupyter/nbdime

7. Build a Pipeline (Experimentation -> Production)

- Generalize notebooks for reusability
  - Accept different input data
  - Different versions of packages
- Cleanly organized. From the top:
  - All imports
  - Key variable declarations
  - Preparatory steps – e.g. – data cleaning
- Re-run entire notebook in one shot to catch bugs
- Parametrize notebooks (e.g. – Papermill)
  - Run via command line / interactively
- Link with Makefile – noninteractive execution
- Write tests

8. Share and Explain your Data

- Notebook without data can be useless
- Prepare data
  - Anonymize sensitive data
  - Abridge very large datasets
  - Open format
- Share data (or portion of) publicly
  - Figshare (https://figshare.com/)
  - Zenodo (https://zenodo.org/)
  - Get Digital Object Identifier (DOI) to uniquely identify data

9. Enable Notebooks to be Read, Run, Explored

- Share notebook on GitHub or similar
- Add license info (reuse)
- Add supplements:
  - ReadMe
  - Static HTML / PDF exports of completed notebooks
    - What it will look like if everything worked perfectly
  - Alternatively view through NbViewer
  - Upload / link to related academic articles
- Zero-install environments (containers)
  - Binder
  - Docker

10. Contribute to Reproducible & Open Research

- Make reproducibility part of the standard process
  - Not an afterthought
- Crowdsource testing / readability
- Consider:
  - Open-sourcing code
  - Contributing to other projects

Binder

Enter your repository information

Provide in the above form a URL or a GitHub repository that contains Jupyter notebooks, as well as a branch, tag, or commit hash. Launch will build your Binder repository. If you specify a path to a notebook file, the notebook will be opened in your browser after building.

We build a Docker image of your repository

Binder will search for a dependency file, such as requirements.txt or environment.yml, in the repository's root directory (more details on more complex dependencies in documentation). The dependency files will be used to build a Docker image. If an image has already been built for the given repository, it will not be rebuilt. If a new commit has been made, the image will automatically be rebuilt.

Interact with your notebooks in a live environment!

A JupyterHub server will host your repository's contents. We offer you a reusable link and badge to your live repository that you can easily share with others.

Teaching with Notebooks before Binder

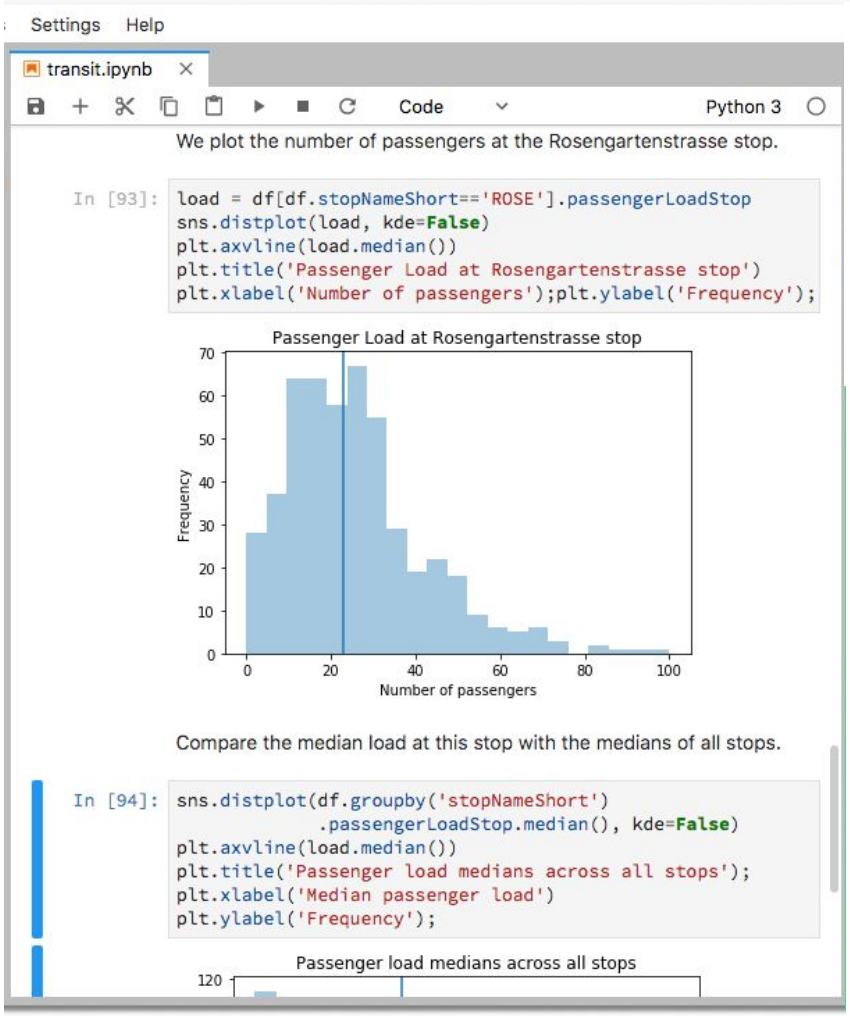
- Before the workshop:
  - Write clean notebooks
  - Try notebooks on multiple configurations manually!
  - Write good install instructions
  - Hope attendees will install requirements
- At the workshop:
  - 50 – 75 % attendees have not installed software
  - Slow conference Wi-Fi!
  - Someone is running some untested configuration
    - Struggle with installation. Repeat.
  - Schedule off by an hour!

Teaching with Notebooks using Binder

- Before the workshop:
  - Write clean notebooks
  - ~~Try notebooks on multiple configurations manually!~~
  - Configure Binder with `requirements.txt` etc.
  - ~~Hope attendees will install requirements~~
- At the workshop:
  - Point everyone towards the Binder link
  - Everyone is up and running in < 5 mins
- Other options – Azure Notebooks, Google Collaboratory, ….

Jupyter Lab - Notebook on steroids

Live-updating visualization panel



Customizable tab areas

Terminals, scripts, notebooks,…

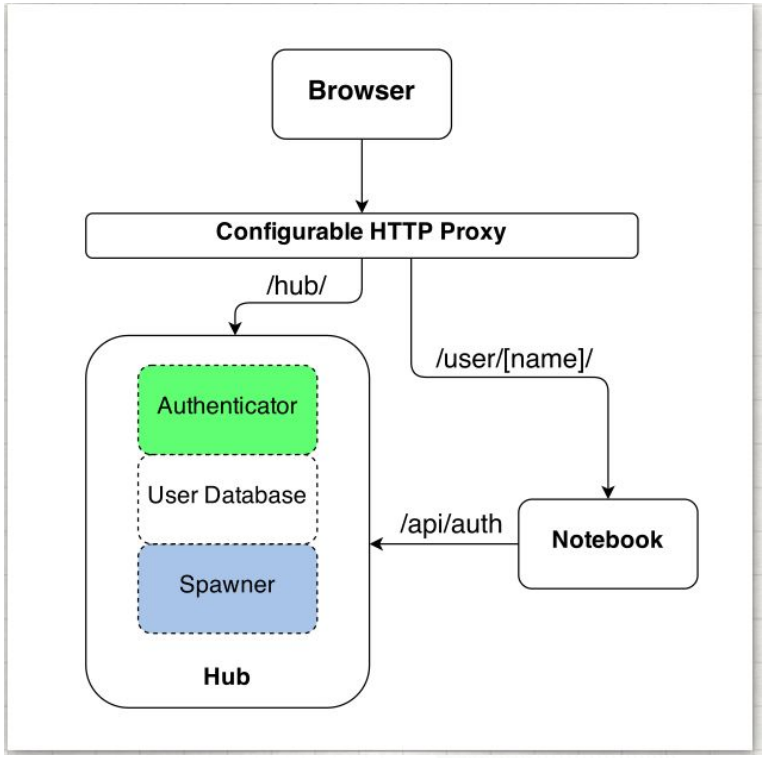
Native file-editors with live-preview

Jupyter Lab - Notebook on steroids

- Drag-and-drop to reorder notebook cells and copy them between notebooks.
- Run code blocks interactively from text files (.py, .R, .md, .tex, etc.).
- Link a code console to a notebook kernel to explore code interactively without cluttering up the notebook with temporary scratch work.
- https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html

Jupyter Hub -> Multi-user notebook server

- Standardized + managed environment
  - Users don't have to install anything
  - Access server via simple webpage
- Standardized authentication (e.g. – Google)
- Optionally – make compute + data available to all
- Great for classes, labs



Jupyter + Cluster / HPC

Web Browser

docker

JupyterHub Web Server

--qos=interactive

NERSC

Cori Login Node

Notebook Server Process

Kernel Process

Cori Compute Node

Notebook Server Process

Kernel Process

Cori Compute Node

Notebook Server Process

Cori Compute Node

Kernel Process

https://www.exascaleproject.org/event/jupyter/

Summary

- Jupyter notebooks - interactive web documents with code, markdown, results, plots, interactive widgets
- Great for:
  - Computational narrative tool
  - Exploratory programming
- Bad for:
  - Out-of-order / deleted cells - hidden states
  - Does not address reproducibility
  - Poor modularity and reusability
- 10 best-practices
- Beyond Notebooks:
  - Jupyter Lab - customizable panels, code editors, etc.
  - Jupyter Hub - Multi-user notebooks
  - Links to high-performance computation

Thank you!

somnaths@ornl.gov

https://docs.google.com/presentation/d/1hZjKeu2X6vorHZxyQE0PDMl8bts3iNXxf1e7DG8X3h4/edit?usp=sharing