Sandia National Laboratories

Exceptional service in the national interest

SCIENTIFIC MACHINE LEARNING AND TENSORFLOW TUTORIAL

Introduction and TensorFlow basics

Ravi G Patel

Scientific Machine Learning Department

February 1 – 2, 2024

Numerical PDEs: Analysis, Algorithms, and Data Challenges

ICERM

Brown University

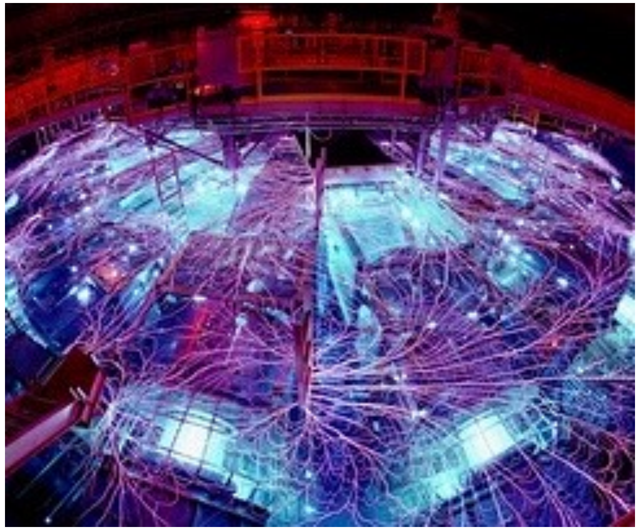
U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2024-00806O

THE GOAL OF SCIENTIFIC MACHINE LEARNING

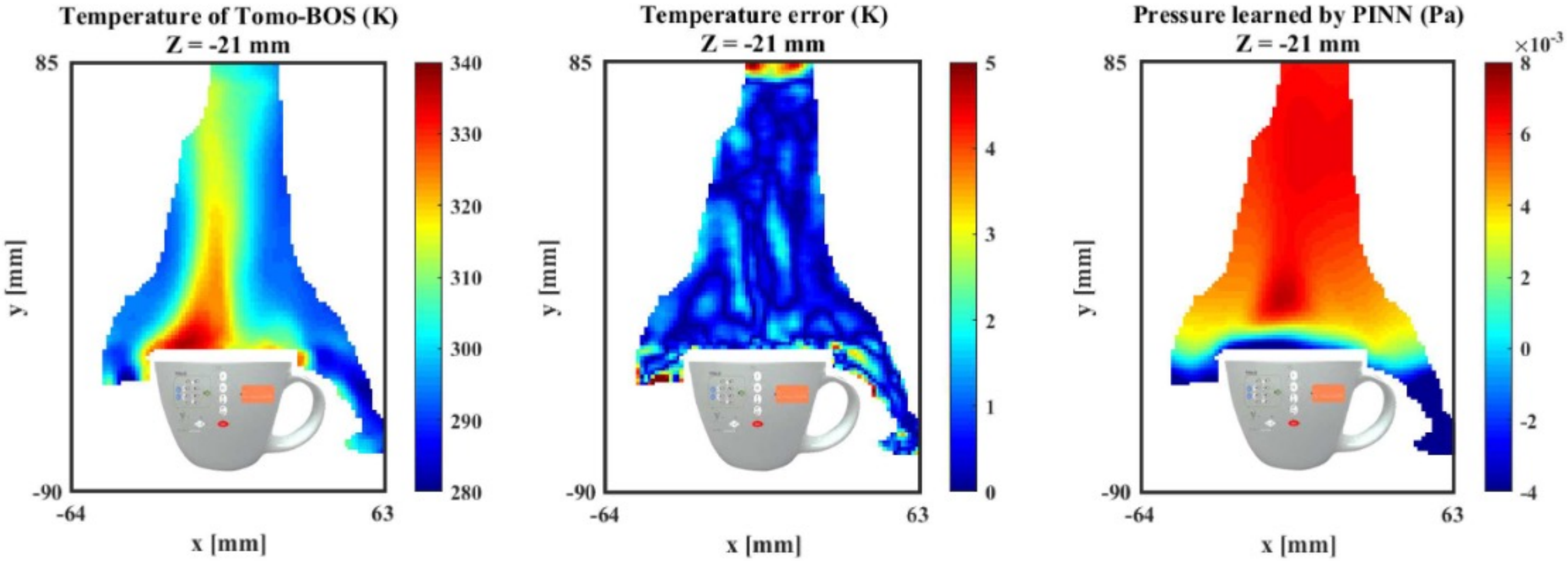


- Given experimental/high fidelity simulation data from a system,
- Find a mathematical model that describes the system
- Experiments/simulations generate **noisy, biased, sparse** data

WHERE IS SCIENTIFIC MACHINE LEARNING USED?

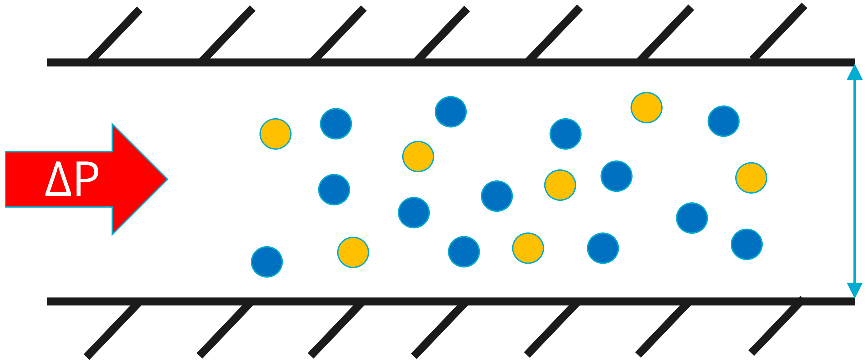
PDE Inverse problems

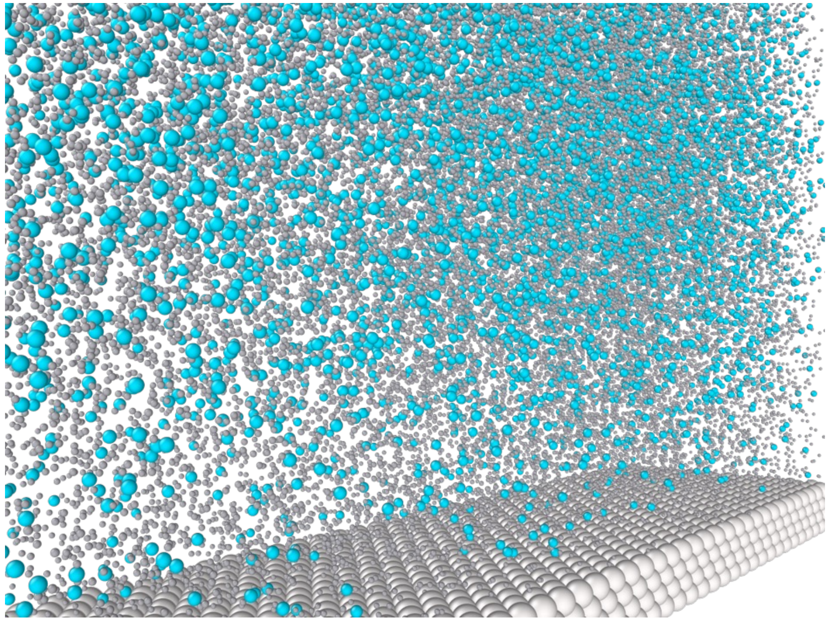


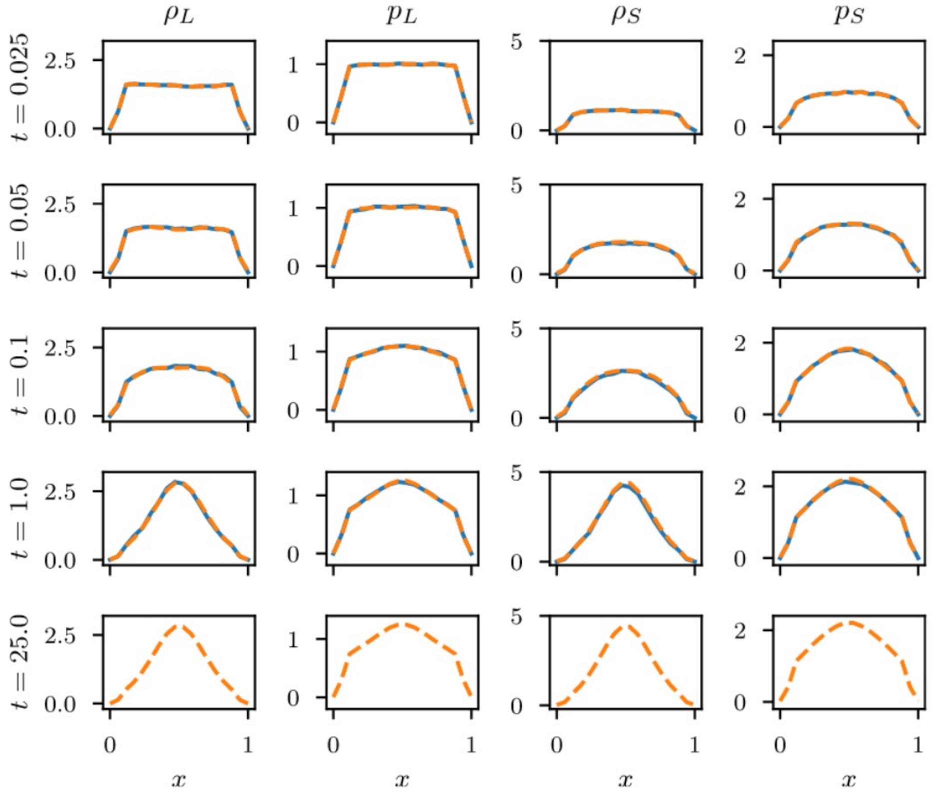
PINNs infers the Pressure of the flow over a coffee mug
from Tomographic background oriented schlieren images
S. Cai et al., *JFM* (2021)

WHERE IS SCIENTIFIC MACHINE LEARNING USED?

Surrogate modeling



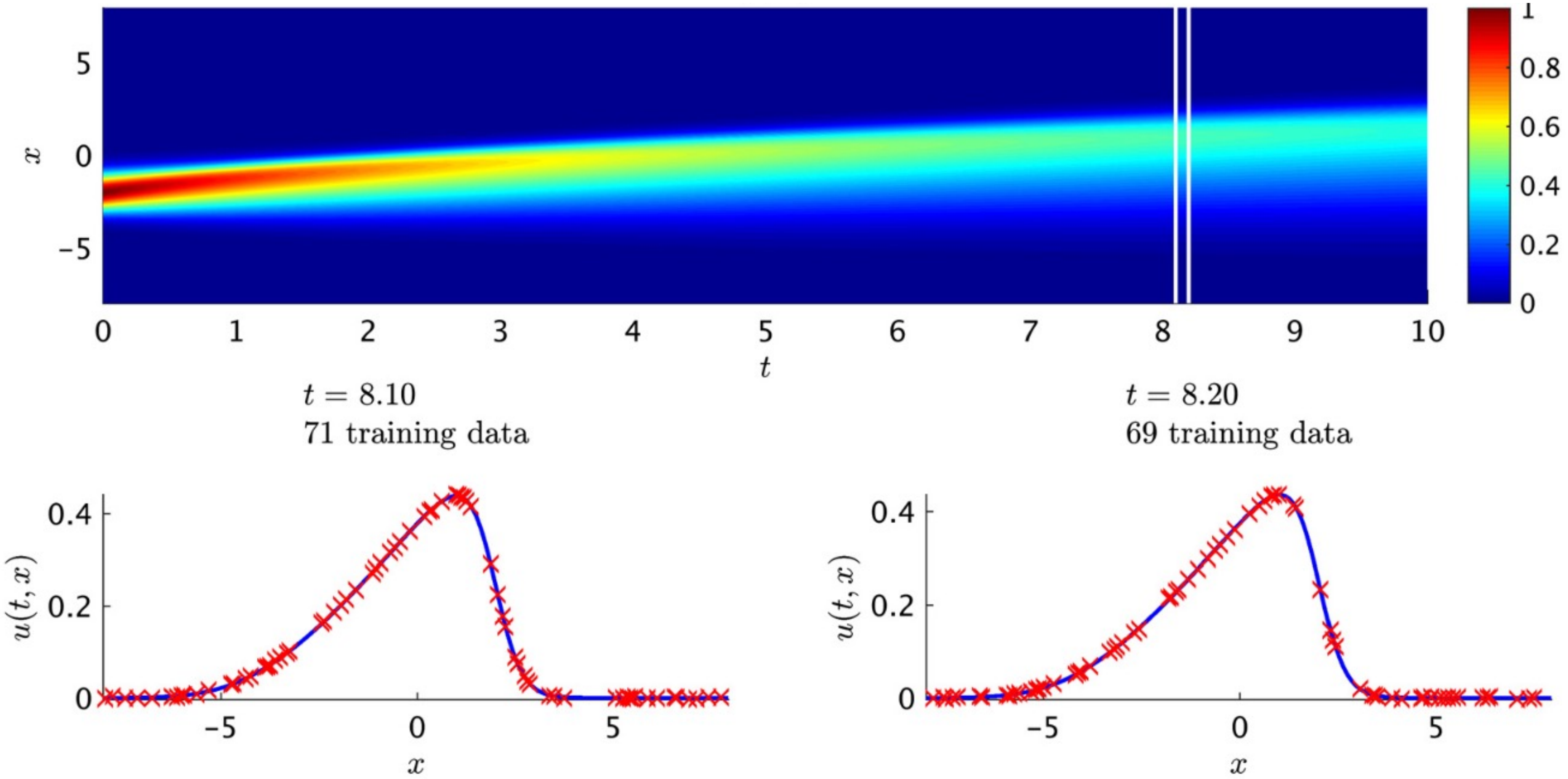




MOR-physics learns dynamics of colloidal system from molecular dynamics simulations
R. Patel et al., *CMAME* (2021)

WHERE IS SCIENTIFIC MACHINE LEARNING USED?

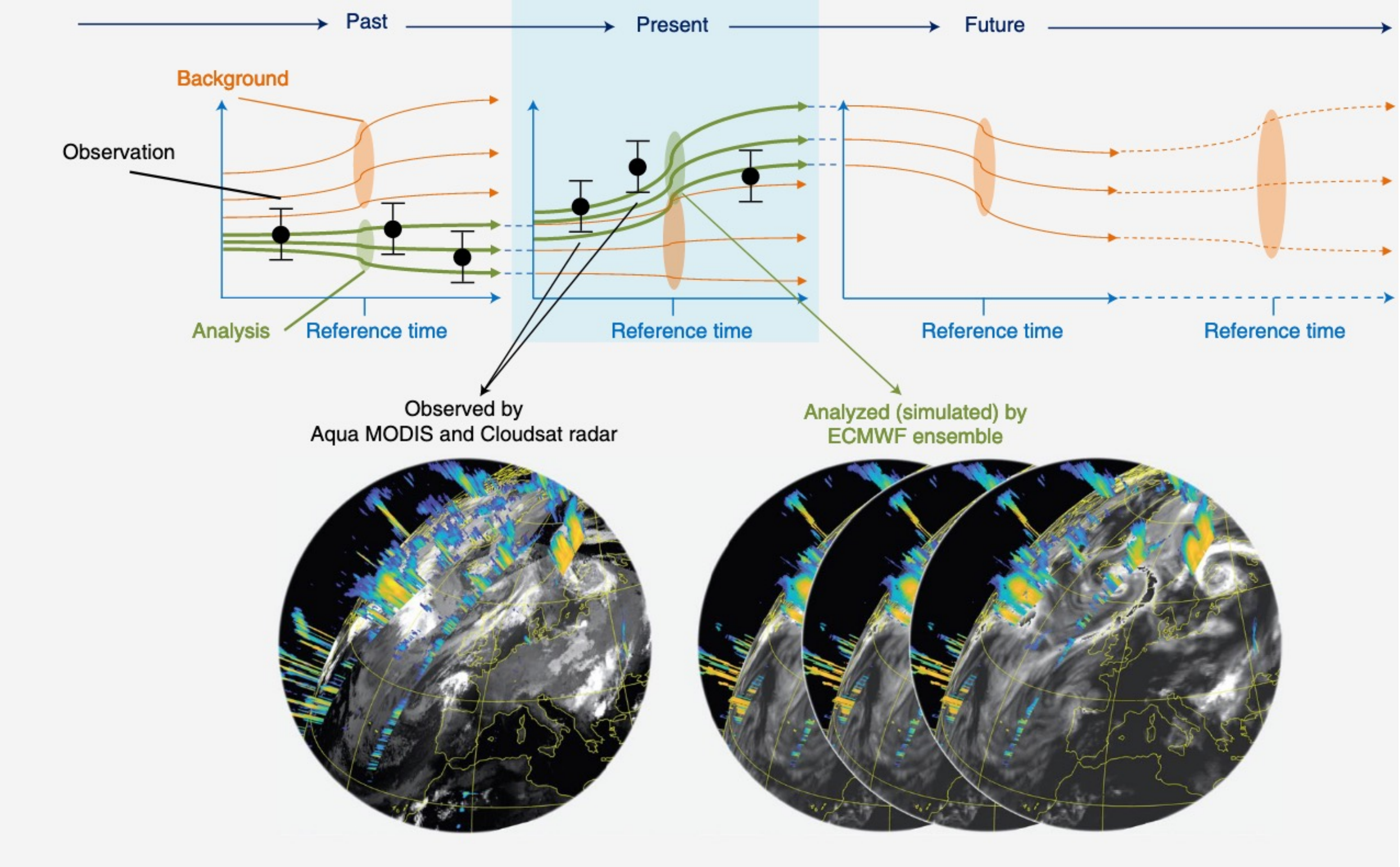
System Identification



A Gaussian process is used to recover a PDE from data
M. Raissi and G. Karniadakis, *JCP* (2018)

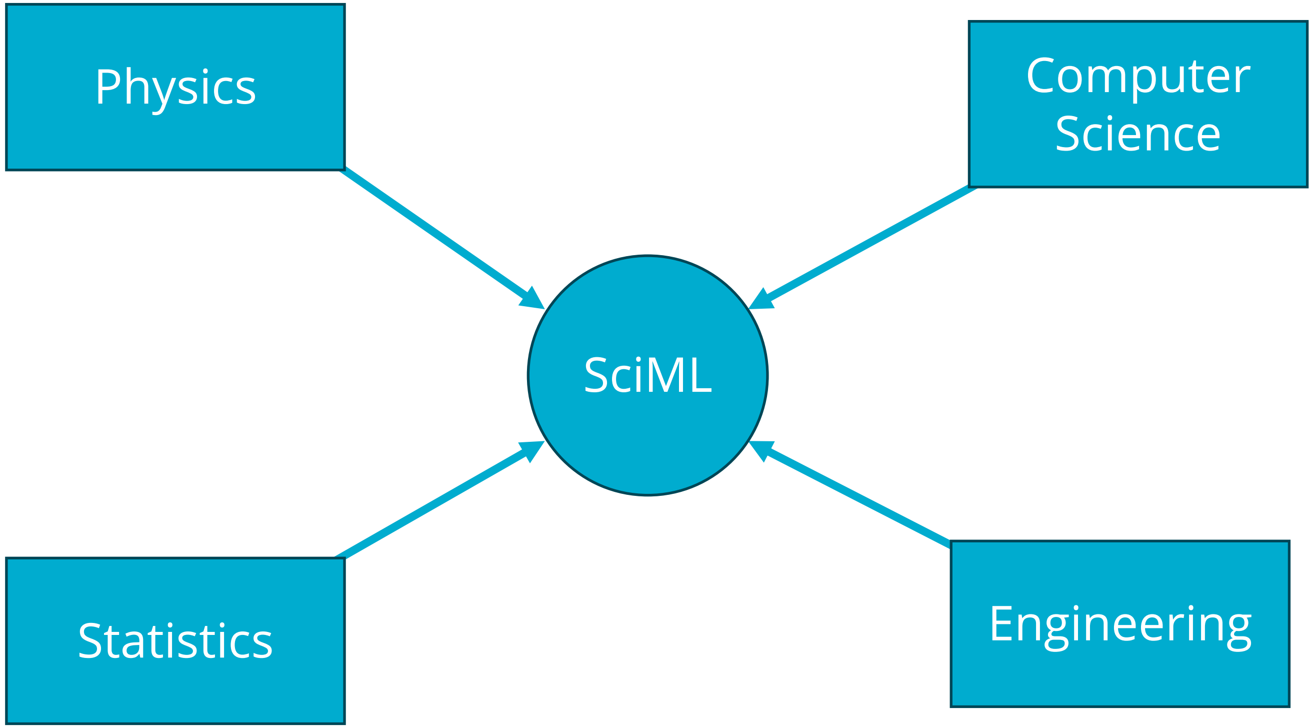
WHERE IS SCIENTIFIC MACHINE LEARNING USED?

Digital Twins

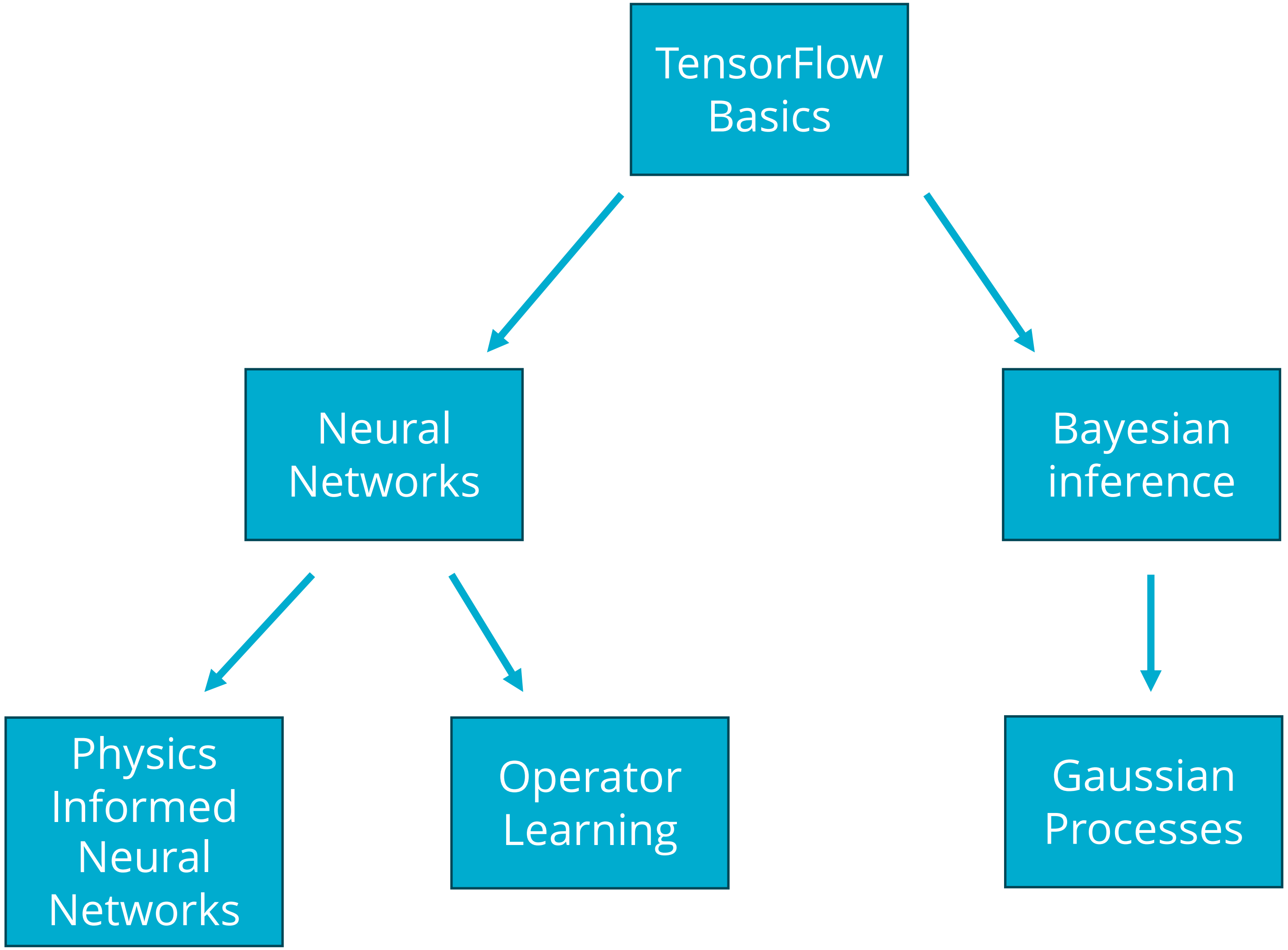


An Earth digital twin - combining MODIS and Cloudsat observations with ECMWF simulations
Bauer et al., *Nature Computational Science*, 2021

COMPONENTS OF SCIENTIFIC MACHINE LEARNING



TUTORIAL TOPICS



SCHEDULE

Day 1

Time	Topic
9:00am – 9:45am	Introduction and TensorFlow basics
9:45am – 10:15am	Neural networks I
10:15am – 10:30am	Coffee Break
10:30am – 11:30am	Neural networks II
11:30am – 1:30pm	Lunch/Free Time
1:30pm – 4:00pm	Physics informed neural networks and inverse problems
4:00pm – 4:30pm	Coffee Break
4:30pm – 5:00pm	Bayesian inference and Gaussian Processes I

SCHEDULE

Day 2

Time	Topic
9:00am – 10:15am	Bayesian Inference and Gaussian Processes II
10:15am – 10:30am	Coffee Break
10:30am – 11:30am	Operator Learning I
11:30am – 1:30pm	Lunch/Free Time
1:30pm – 3:30pm	Operator Learning II
3:30pm – 4:00pm	Coffee Break
4:00pm – 4:30pm	Advanced topics. Future directions. Reproducibility.
4:30pm – 5:00pm	Open Discussion

TENSORFLOW OVERVIEW

- Tensorflow – linear algebra with automatic differentiation and accelerator support
- Concepts
  - Tensor data type
  - Accelerators
  - GradientTape
- Addons
  - Keras
  - Optimizers
  - Tensorflow probability

TENSORFLOW VS. NUMPY

Numpy

- `import numpy as np`
- Operates on np.array type
- np.zeros, np.einsum, np.arange, np.shape
- np.concat, np.sum, A.T

Tensorflow

- `import tensorflow as tf`
- Operates on the tf.tensor type
- tf.zeros, np.einsum, tf.arange, tf.shape
- tf.concatenate, tf.reduce_sum, tf.transpose(A)
- Accelerator support
- Automatic differentiation

TENSORFLOW EXAMPLE CODE

TensorType

- Similar to numpy arrays, but with accelerator support and automatic differentiation
- Constants once instantiated cannot be modified
- ```
  In [1]: tf.constant(4.)
  Out[1]: <tf.Tensor: shape=(), dtype=float32, numpy=4.0>
  ```
- Variables (i.e. parameters) can be reassigned values
- ```
  In [2]: a = tf.Variable(4.)
  In [3]: a.assign(3.)
  Out[3]: <tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=3.0>
  ```
- The standard `=` operator will create a new tensor
- Accumulation operators like `+=` will give an error
- For optimization, we typically will use `assign` to update Variables (i.e. parameters)

TENSORFLOW EXAMPLE CODE

GradientTape

- Compute gradients using the following syntax,

```
x = tf.constant(2.)
with tf.GradientTape() as tape:
    tape.watch(x)
    y = x**2
tape.gradients(y,x)
```

- GradientTape tracks operations and adjoints for backprop
  - More details in next topic
- Can only compute the gradient of a scalar
  - Will sum over first argument if it's not a scalar
- GradientTape can also be used to compute Hessians and Jacobians
  - https://www.tensorflow.org/guide/advanced_autodiff

TENSORFLOW EXAMPLE CODE

Eager mode

- TensorFlow offers two computational modes
- Eager mode is the default mode
  - Behaves much like standard Python but is much slower
  - Easier to debug
  - Code so far has all been in eager mode

```
A = tf.random.normal((25,250,2500))
B = tf.random.normal((2500,250,25))
def DoubleDot(A,B):
    return tf.einsum('ijk,kjl',A,B)
```

```
In [1]: %%timeit
   ...: DoubleDot(A,B)
8.54 ms ± 192 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

TENSORFLOW EXAMPLE CODE

Graph mode

- Graph mode is much faster but more restrictive. To invoke Graph mode
  1. Write a Python function with TensorFlow operations
  2. Pass the function through `tf.function` or decorate one with `@tf.function`
  3. First run will be slow

```
DoubleDot_graph = tf.function(DoubleDot)

@tf.function(jit_compile=True)
def DoubleDot_graph2(A,B):
    return tf.einsum('ijk,kjl',A,B)
```

```
In [1]: %%timeit
   ...: DoubleDot_graph(A,B)
1.11 ms ± 146 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

In [2]: %%timeit
   ...: DoubleDot_graph2(A,B)
1.11 ms ± 168 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

TENSORFLOW EXAMPLE CODE

Just-in-time compilation

- We can further speed up TensorFlow code by specifying just-in-time compilation (XLA) in tf.function,

```
@tf.function(jit_compile=True)
def DoubleDot_jit(A,B):
    return tf.einsum('ijk,kjl',A,B)
```

```
In [1]: %%timeit
   ...: DoubleDot_jit(A,B)
260 µs ± 269 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

TENSORFLOW EXAMPLE CODE

Executing on accelerators

- By default, TensorFlow will try to execute code on the GPU
- We can explicitly tell it to execute on a specific device,

```
In [1]: %%timeit
   ...: with tf.device('/CPU:0'):
   ...:     DoubleDot_jit(A,B)
8.35 ms ± 80.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [2]: %%timeit
   ...: with tf.device('/GPU:0'):
   ...:     DoubleDot_jit(A,B)
262 µs ± 319 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

TIPS AND TRICKS

- Eager mode is more robust and provides better error messages than graph mode
  - Check code in eager before switch to graph mode
  - Check code in graph mode without JIT before enabling JIT
- Avoid using numpy code and operations with side effects inside a tf.function
- Use Jupyter's built-in debugger
- Applying operations to the wrong shapes is the most common error
  - Check the shape of inputs to/outputs of operations
- TensorFlow is much less forgiving than numpy with respect to precision
  - By default float32 vs. numpy's float64
- GradientTape must be used with tf.tensor. np.array's won't trace
- You'll likely do some tensor transposing and reshaping,

  e.g., to use linear algebra operations, $\mathbb{R}^{n \times m} \cong \mathbb{R}^{nm}$
- TensorFlow and numpy are not one-to-one. Check the documentation, https://www.tensorflow.org/api_docs

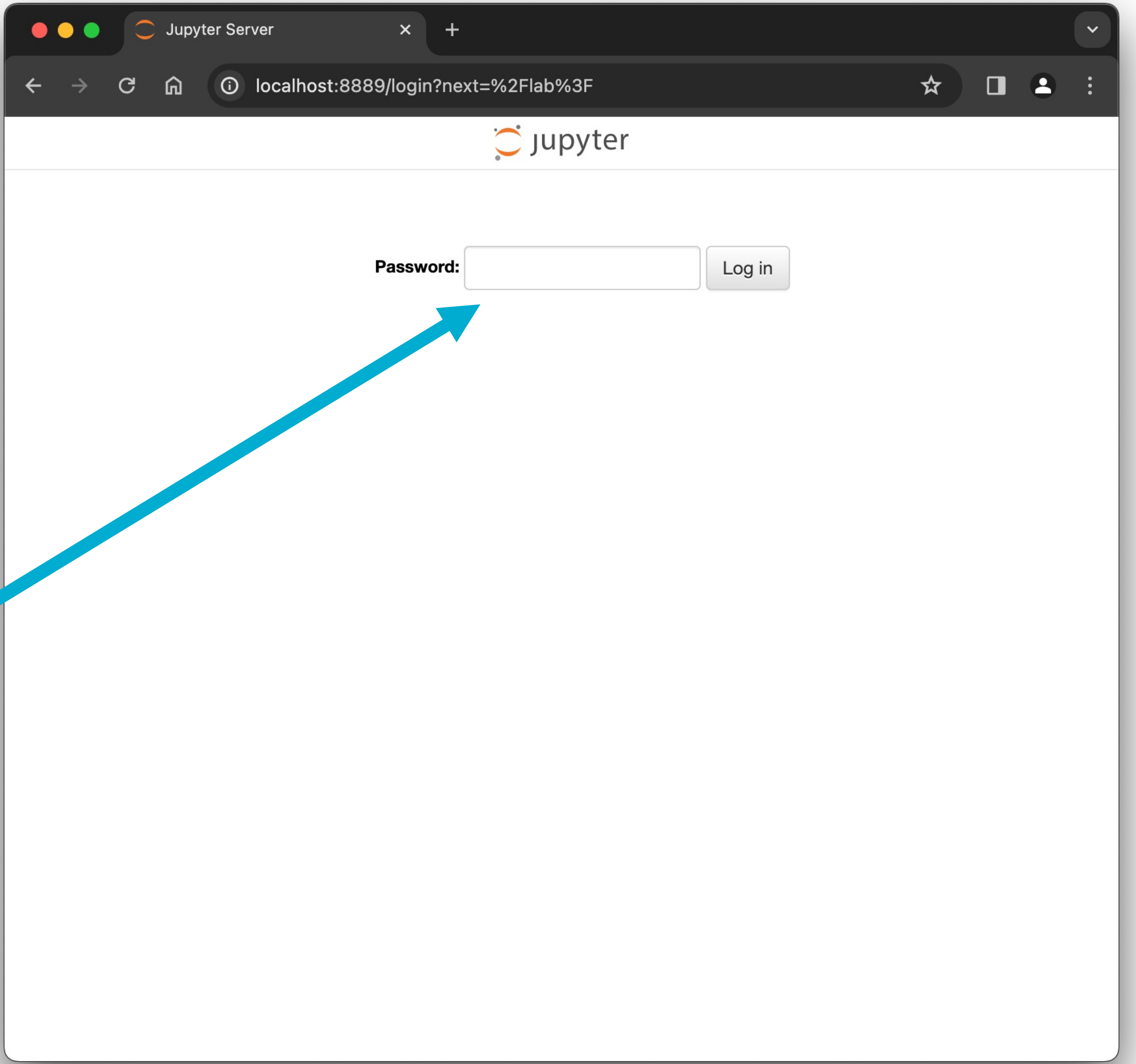
HANDS ON EXERCISES

- We'll split up into 10 groups, each with an assigned instance
  - Amazon EC2 with 4 vCPUs and 1 Nvidia T40 GPU
  - Only up until tomorrow night – save data offline
- We'll be using the Jupyter lab to do the exercises
  - Web browser-based development environment
  - Collaborative – group members will share same Jupyter
- The instances have all been started already
- Run the following command in your terminal,

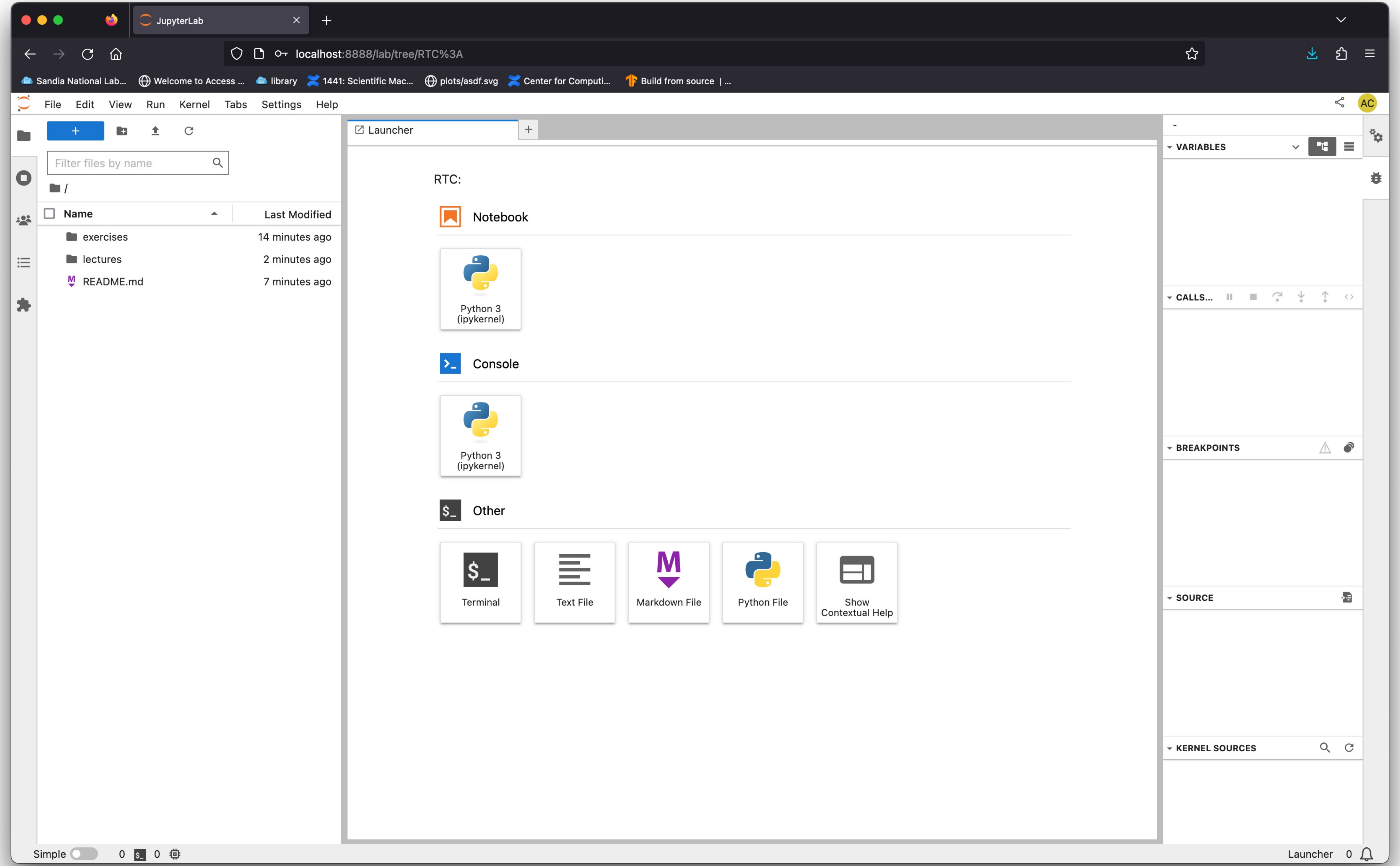
```
$ ssh -L 8888:localhost:8888 icerm2@<ip>
```

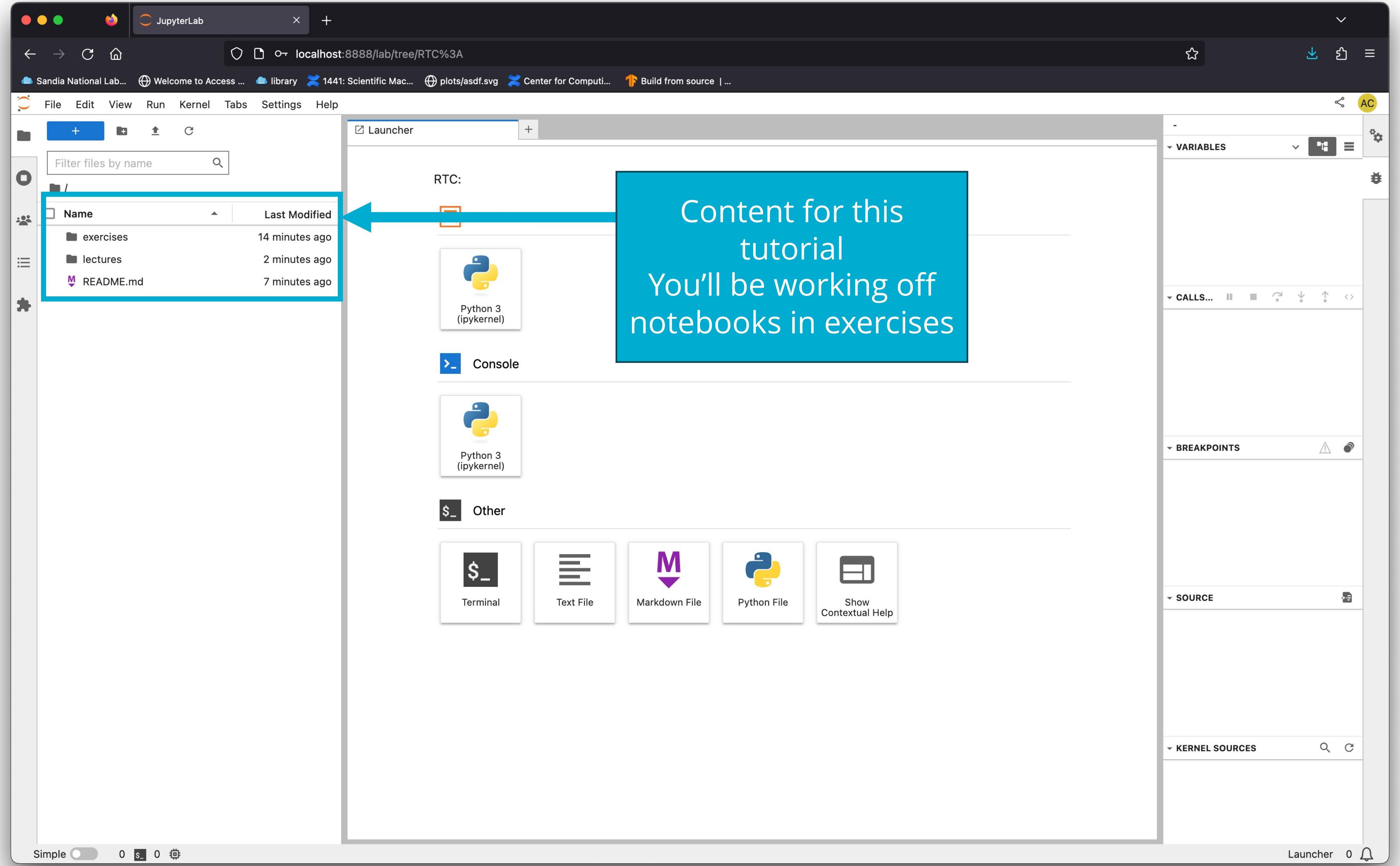
  - I'll give each group their `<ip>`
- and in your browser, navigate to
  - http://localhost:8888 (password is icerm2024)

JUPYTER NOTEBOOK

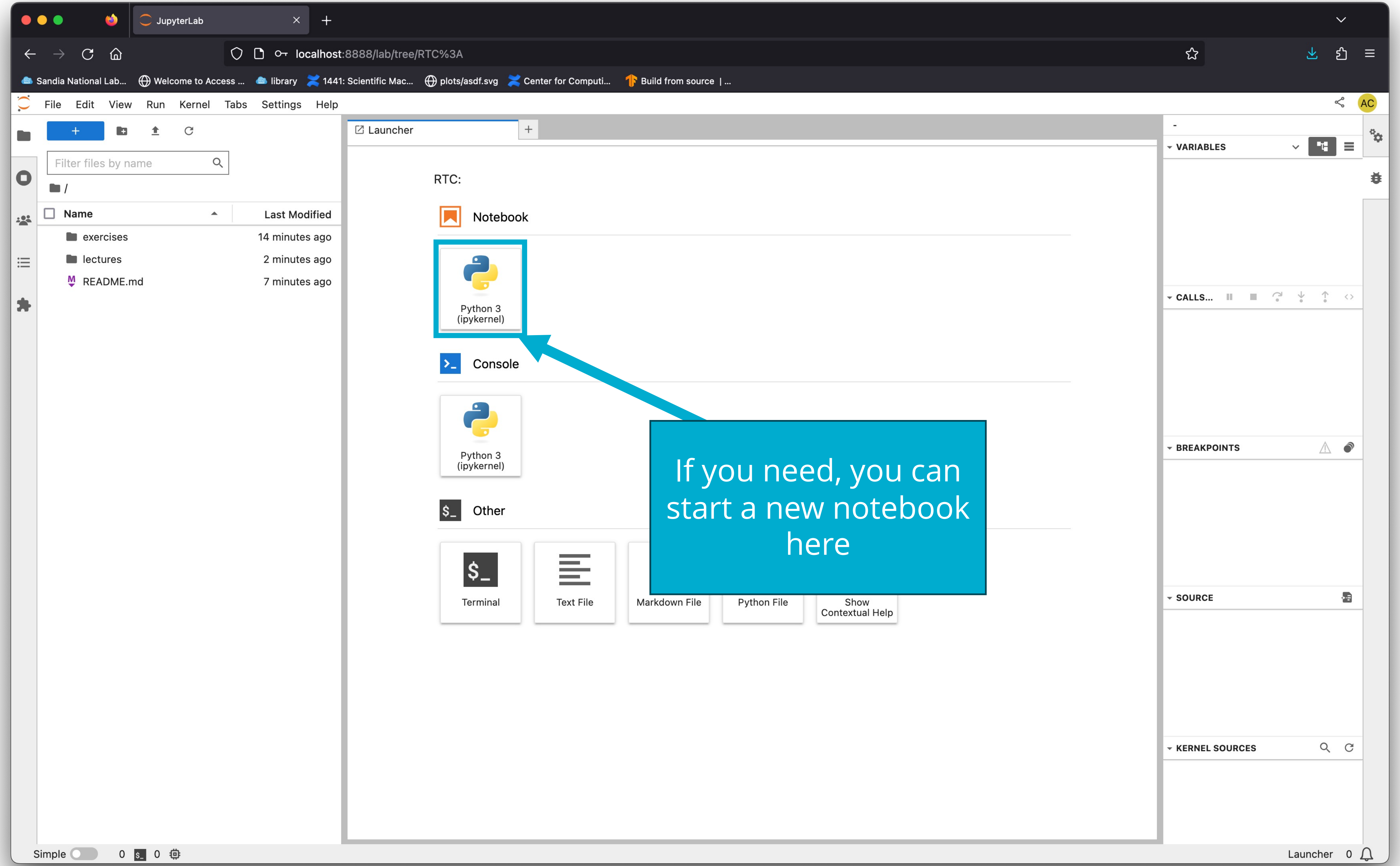


Enter Password:
icerm2024

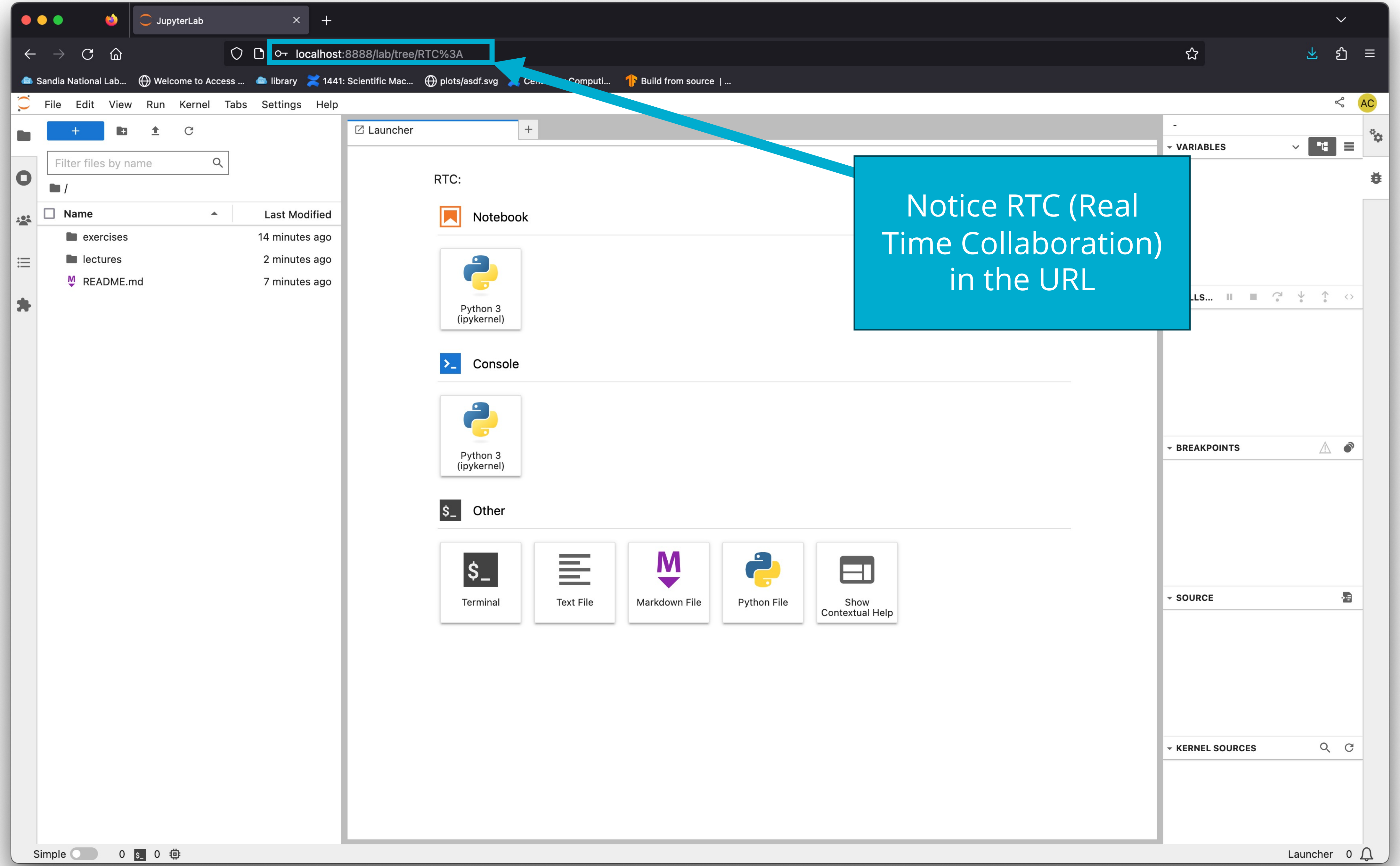

JupyterLab
localhost:8888/lab/tree/RTC%3A
Sandia National Lab...
Welcome to Access ...
library
1441: Scientific Mac...
plots/asdf.svg
Center for Computi...
Build from source | ...
File
Edit
View
Run
Kernel
Tabs
Settings
Help
AC
Filter files by name
/
Name
Last Modified
exercises
14 minutes ago
lectures
2 minutes ago
README.md
7 minutes ago
Launcher
RTC:
Notebook
Python 3 (ipykernel)
Console
Python 3 (ipykernel)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help
VARIABLES
CALLS...
BREAKPOINTS
SOURCE
KERNEL SOURCES
Simple
0
0
Launcher
0



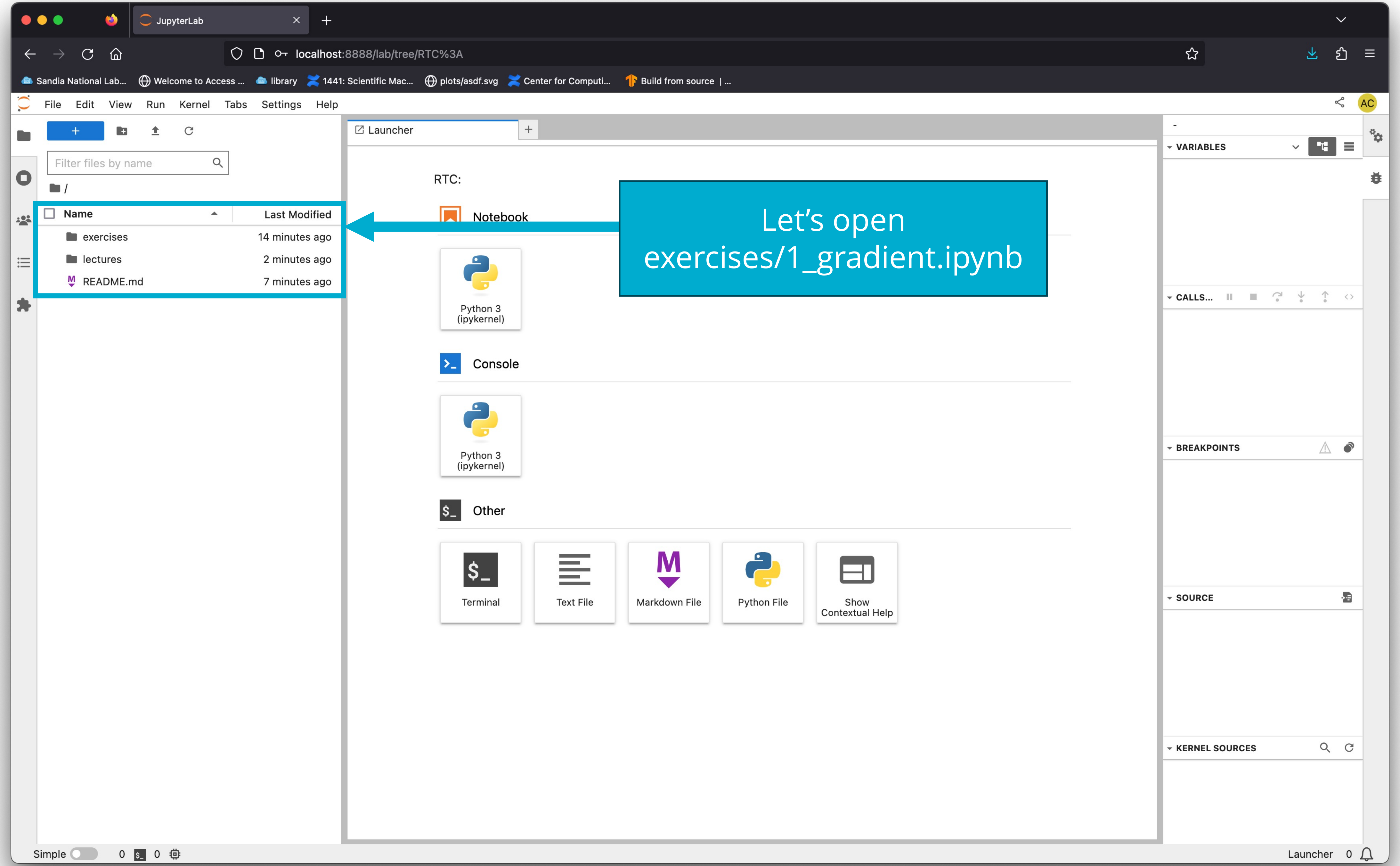
Content for this tutorial
You'll be working off notebooks in exercises



If you need, you can start a new notebook here



JupyterLab
localhost:8888/lab/tree/RTC%3A
File Edit View Run Kernel Tabs Settings Help
Filter files by name
Name
Last Modified
exercises
14 minutes ago
lectures
2 minutes ago
README.md
7 minutes ago
Launcher
RTC:
Notebook
Python 3 (ipykernel)
Console
Python 3 (ipykernel)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help
VARIABLES
BREAKPOINTS
SOURCE
KERNEL SOURCES
Simple
Launcher
Notice RTC (Real Time Collaboration) in the URL



Let's open exercises/1_gradient.ipynb

EXERCISE

- Follow the previous slides to open the exercises/1_gradient.ipynb notebook
- Compute the gradient of Ax for a random matrix, A, and vector, x
- Experiment with tf.function, jit, and running on the CPU/GPU
- Experiment with Jupyter's debugger