Integrating Interactive Jupyter Notebooks at the BNL SDCC

D. Allan, D. Benjamin*, M. Karasawa, K. Li, O. Rind, W. Strecker-Kellogg
Brookhaven National Laboratory, *Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center

BNL Scientific Data & Computing Center (SDCC)

- Located at Brookhaven National Laboratory on Long Island, NY — Largest component of the Computational Science Initiative (CSI)
- Serves an increasingly diverse, multi-disciplinary user community: RHIC Tier-0, US ATLAS Tier-1 and Tier-3, Belle-II Tier-1, Neutrino, Astro, LQCD, NSLS-II, CFN, sPHENIX….more than 2000 users from 20+ projects
- Large HTC infrastructure accessed via HTCondor (plus experiment-specific job management layers)
- Growing HPC infrastructure, currently with two production clusters accessed via Slurm
- Limited interactive resources accessed via ssh gateways

Two modes, Two workflows

- HPC & HTC (parallel vs interlinked, accelerator vs plain-cpu)
  - High-performance systems for GPUs / MPI / accelerators
  - High-throughput systems for big data parallel processing
- Batch & Interactive (working on code/GPUs vs submitting large workflows)
  - Job workflow management
  - Direct development & testing on better hardware

Traditional “Interactive SSH + Batch” paradigm places requirements on the users:

- Must be sufficiently motivated to learn and use batch systems
- Need to buy in to the workflow model: Develop, compile, move data, small-scale run on interactive nodes, full-scale processing on batch

Data Analysis As A Service

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

- New paradigm: **Jupyter Notebooks** (IPython)
  - Expanding the interactive toolset
  - "Literate Computing": Combines code, text, equations within a narrative
  - Easy to document, share, and reproduce results; create tutorials…Lower barrier of entry, both for learning curve and user-base
  - Provides a flexible, standardized, platform independent interface through a web browser
  - Can run with no local software installation
  - Many language extensions (kernels) and tools available

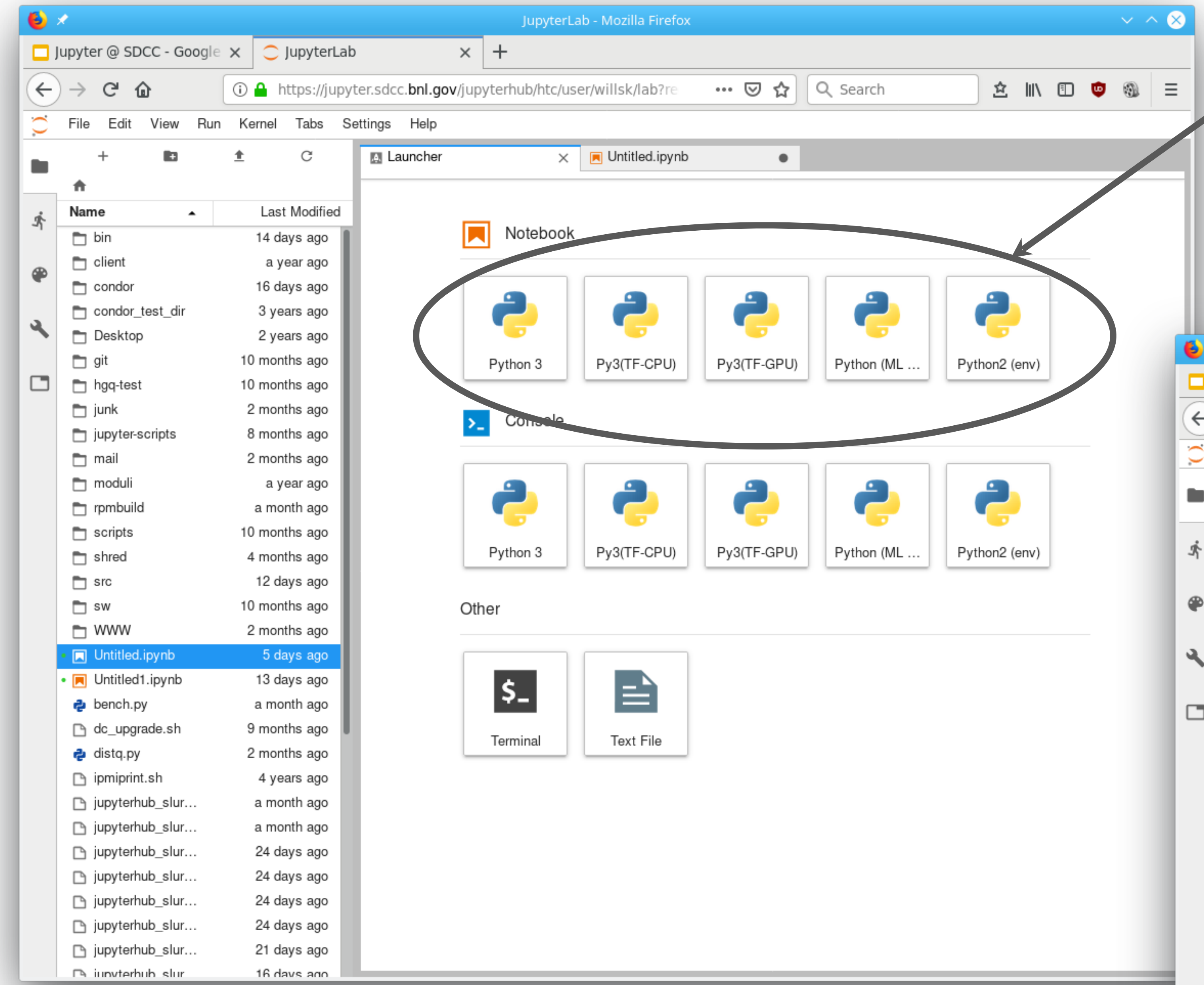
U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center

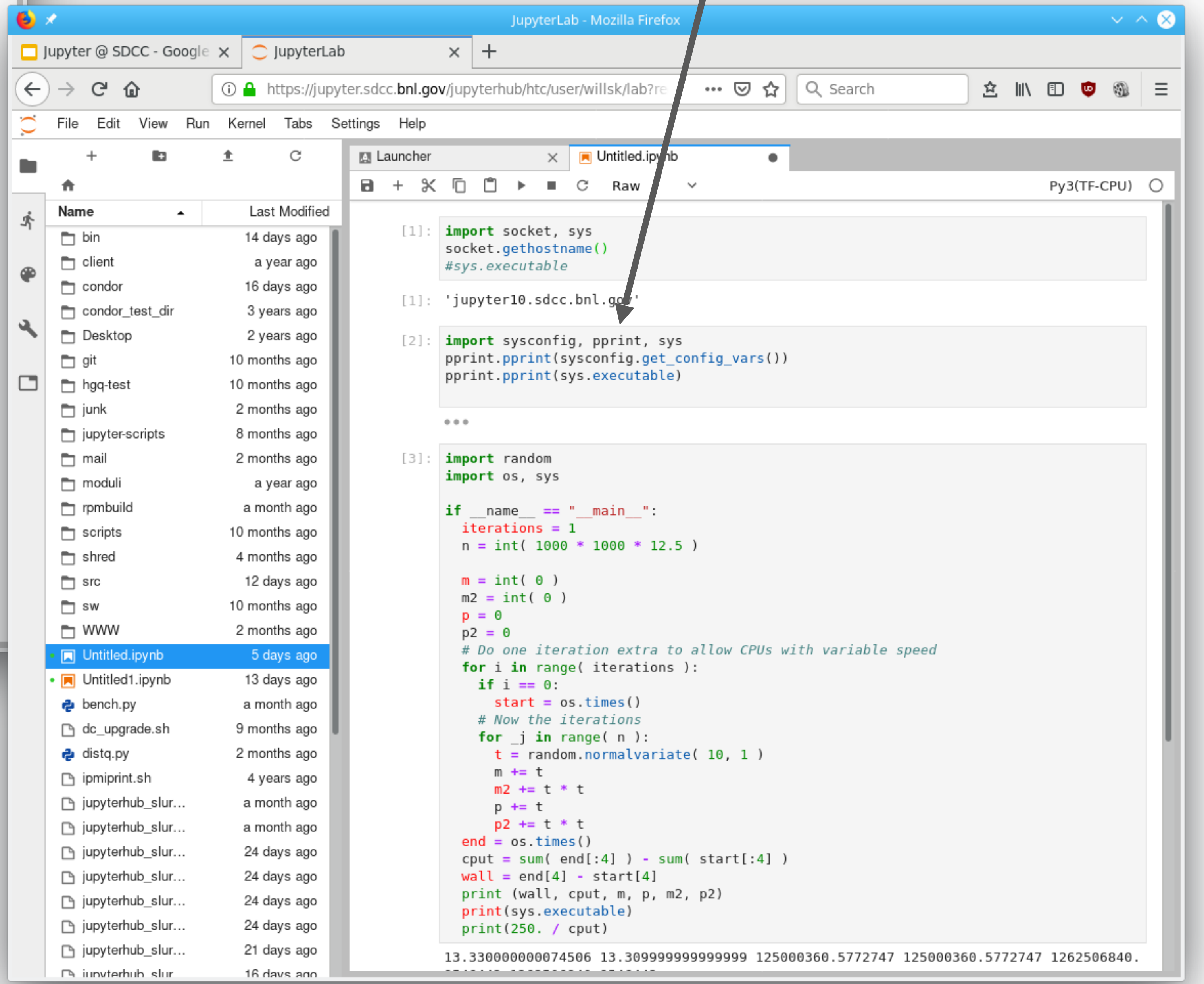
Jupyter Service UI

Kernels

Notebook Documents



Name	Last Modified
bin	14 days ago
client	a year ago
condor	16 days ago
condor_test_dir	3 years ago
Desktop	2 years ago
git	10 months ago
hgq-test	10 months ago
junk	2 months ago
jupyter-scripts	8 months ago
mail	2 months ago
moduli	a year ago
rpmbuild	a month ago
scripts	10 months ago
shred	4 months ago
src	12 days ago
sw	10 months ago
WWW	2 months ago
Untitled.ipynb	5 days ago
Untitled1.ipynb	13 days ago
bench.py	a month ago
dc_upgrade.sh	9 months ago
distq.py	2 months ago
ipmiprint.sh	4 years ago
jupyterhub_slur…	a month ago
jupyterhub_slur…	a month ago
jupyterhub_slur…	24 days ago
jupyterhub_slur…	24 days ago
jupyterhub_slur…	24 days ago
jupyterhub_slur…	24 days ago
jupyterhub_slur…	21 days ago
jupyterhub_slur	16 days ago


JupyterLab - Mozilla Firefox
Jupyter @ SDCC - Google
JupyterLab
https://jupyter.sdcc.bnl.gov/jupyterhub/htc/user/willsk/lab?
Search
File Edit View Run Kernel Tabs Settings Help
Launcher
Untitled.ipynb
Raw
Py3(TF-CPU)

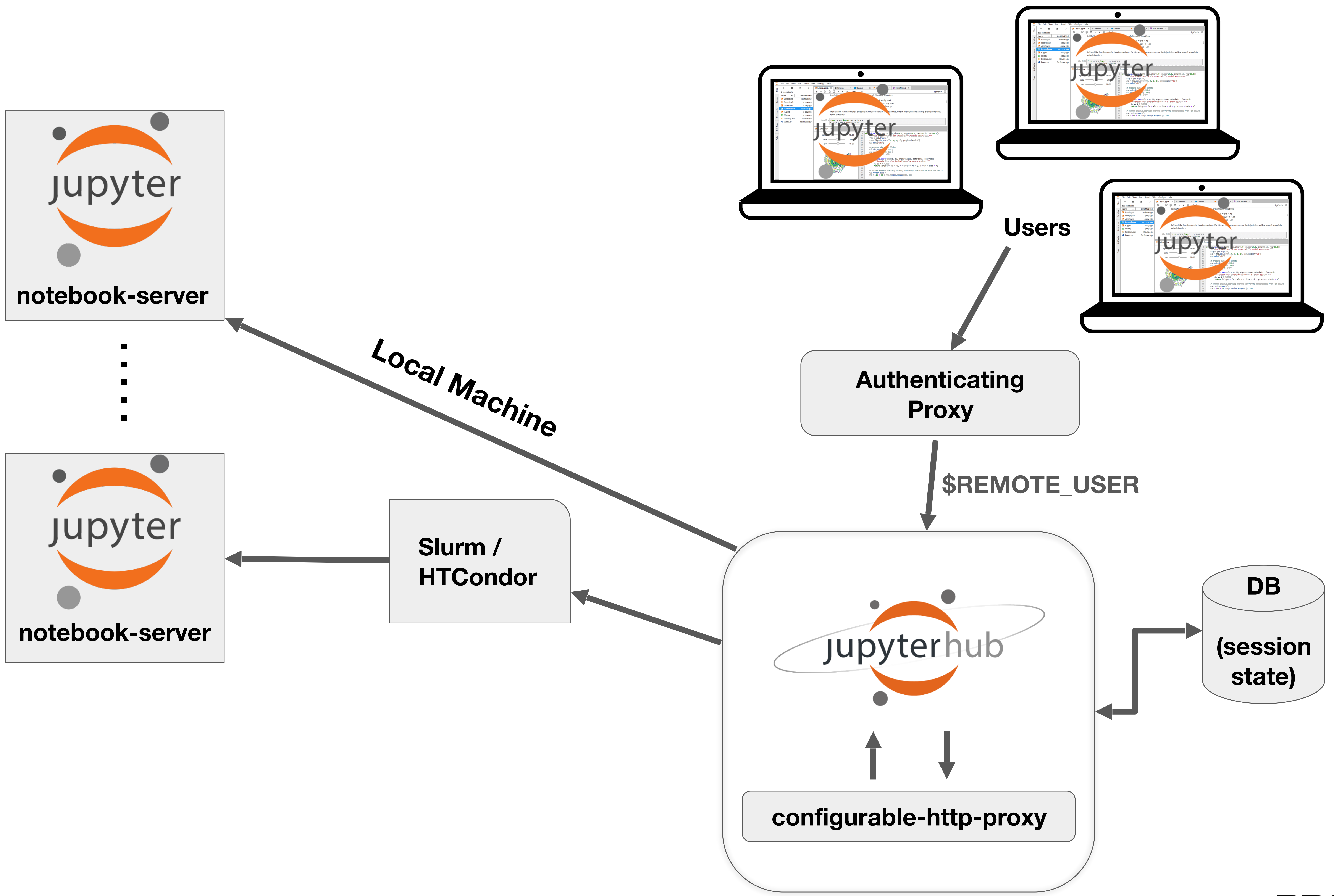
```
[1]: import socket, sys
     socket.gethostname()
     #sys.executable

[1]: 'jupyter10.sdcc.bnl.gov'

[2]: import sysconfig, pprint, sys
     pprint.pprint(sysconfig.get_config_vars())
     pprint.pprint(sys.executable)

     •••

[3]: import random
     import os, sys

     if __name__ == "__main__":
       iterations = 1
       n = int( 1000 * 1000 * 12.5 )

       m = int( 0 )
       m2 = int( 0 )
       p = 0
       p2 = 0
       # Do one iteration extra to allow CPUs with variable speed
       for i in range( iterations ):
         if i == 0:
           start = os.times()
         # Now the iterations
         for _j in range( n ):
           t = random.normalvariate( 10, 1 )
           m += t
           m2 += t * t
           p += t
           p2 += t * t
       end = os.times()
       cput = sum( end[:4] ) - sum( start[:4] )
       wall = end[4] - start[4]
       print (wall, cput, m, p, m2, p2)
       print(sys.executable)
       print(250. / cput)

     13.330000000074506 13.309999999999999 125000360.5772747 125000360.5772747 1262506840.
```

Jupyterlab

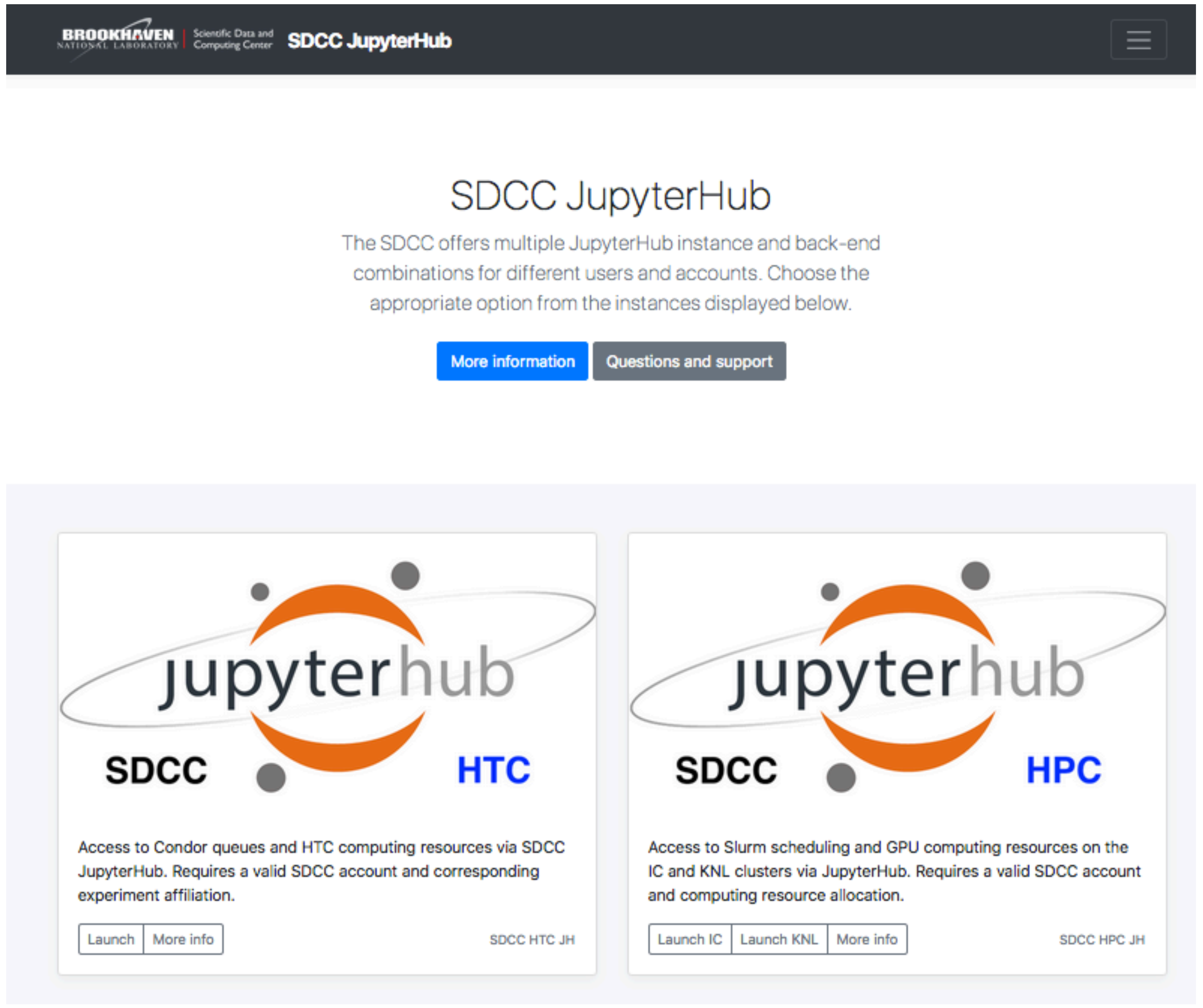
Production Architecture

- Goal: leverage already successful pre-existing resources, expertise, and infrastructure (batch) instead of rolling a new backend service
  - Allow users to leverage any type of computational resource they might need — implies enabling both HTC and HPC/GPU, e.g. upcoming ATLAS ML workflows
- Requirements
  - Expose to the world via unified interface https://jupyter.sdcc.bnl.gov — common solution for HTC and HPC resource access
  - Satisfy cybersecurity constraints
- Design
  - Insert authenticating proxy as frontend to decouple jupyterhub from cybersecurity requirements (e.g. MFA)
  - Scale notebooks via load-balancing as well as via batch systems
    - Automated deployment of multiple hub instances using Puppet
  - Enable access to GPU nodes in a user-friendly way
    - User-specific UI for Slurm spawner support

Jupyterhub Service Architecture



Frontend Proxy Interface


BROOKHAVEN NATIONAL LABORATORY
Scientific Data and Computing Center
SDCC JupyterHub

SDCC JupyterHub

The SDCC offers multiple JupyterHub instance and back-end combinations for different users and accounts. Choose the appropriate option from the instances displayed below.

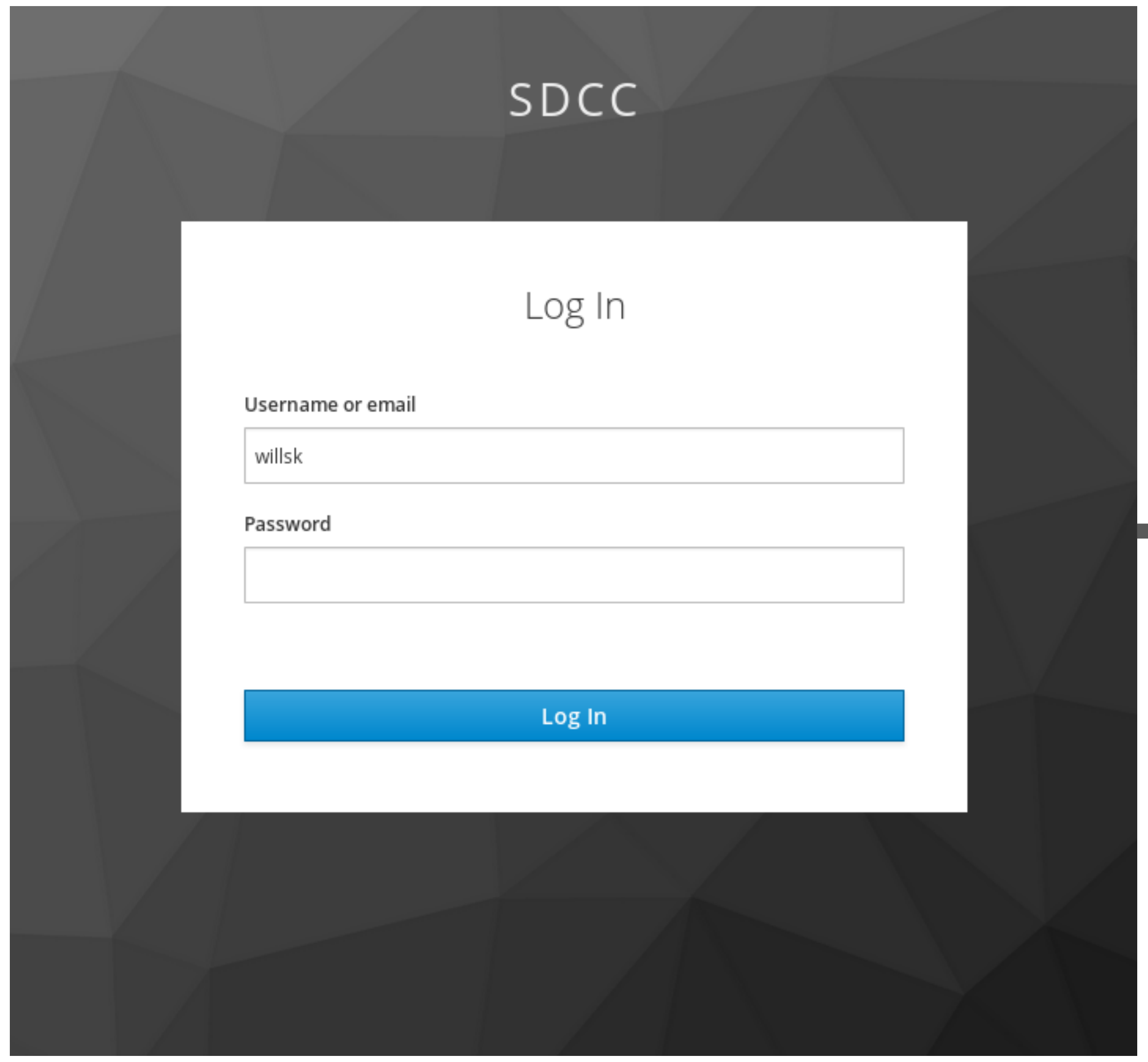
Access to Condor queues and HTC computing resources via SDCC JupyterHub. Requires a valid SDCC account and corresponding experiment affiliation.

Access to Slurm scheduling and GPU computing resources on the IC and KNL clusters via JupyterHub. Requires a valid SDCC account and computing resource allocation.

- For Orchestration: a small cluster of directly-launched jupyter instances
  - HTTP-level Load-balanced from frontend proxy
  - One each on IC and HTCondor shared pool
- For Develop and Test: Use existing batch systems
  - HTCondor and Slurm support running a jupyterlab session as a batch job
  - Containers can enter at batch level to isolate external users or can be based on choice of environment
  - Best way to ensure exclusive, fair access to scarce resources (e.g. GPUs)
  - Open questions: Latency, Cleanup, Starvation

Multifactor Auth

- Using Keycloak MFA tokens
- Google Authenticator or FreeOTP app
- Easy setup by scanning QR code first time



SDCC

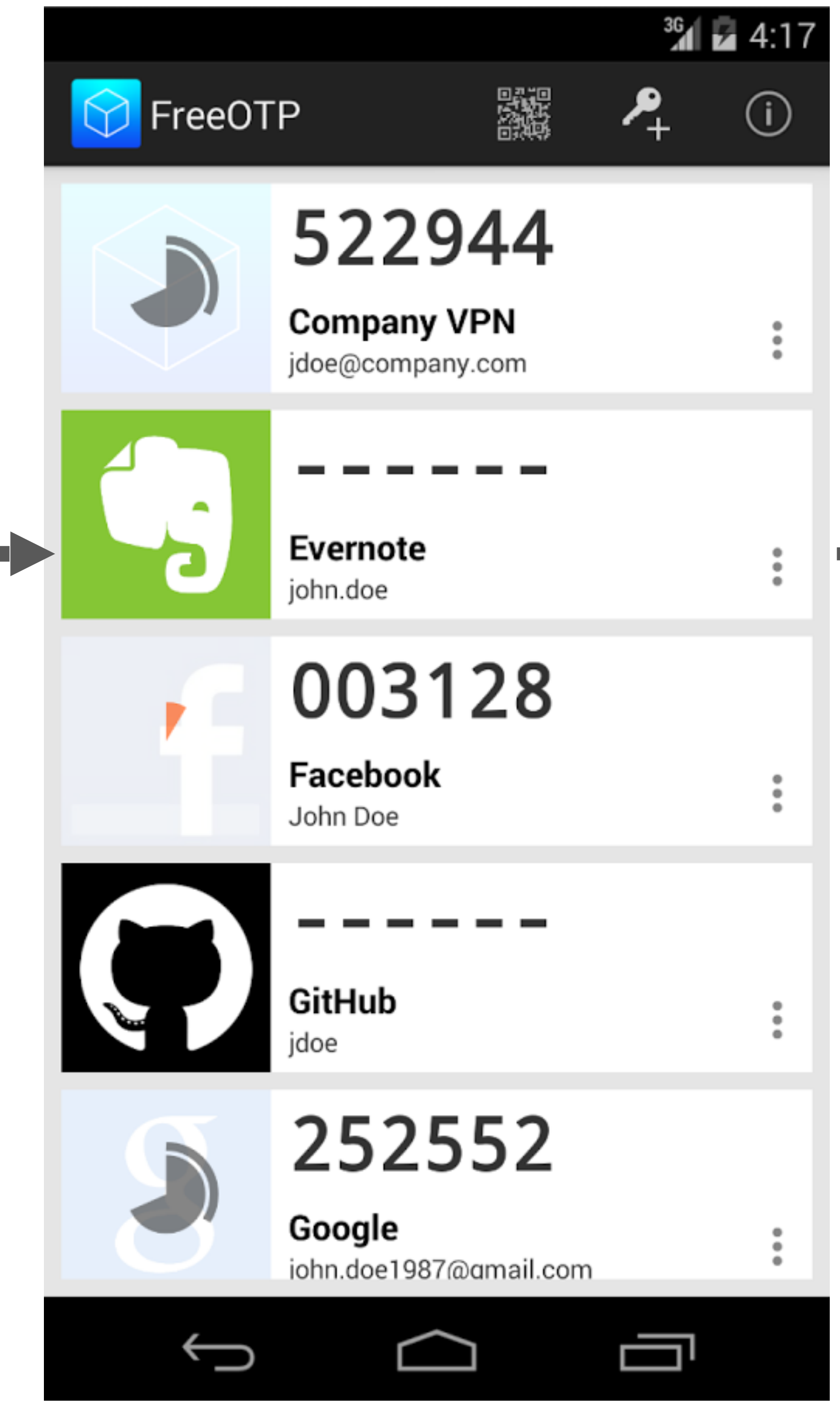
Log In

Username or email

willsk

Password

Log In



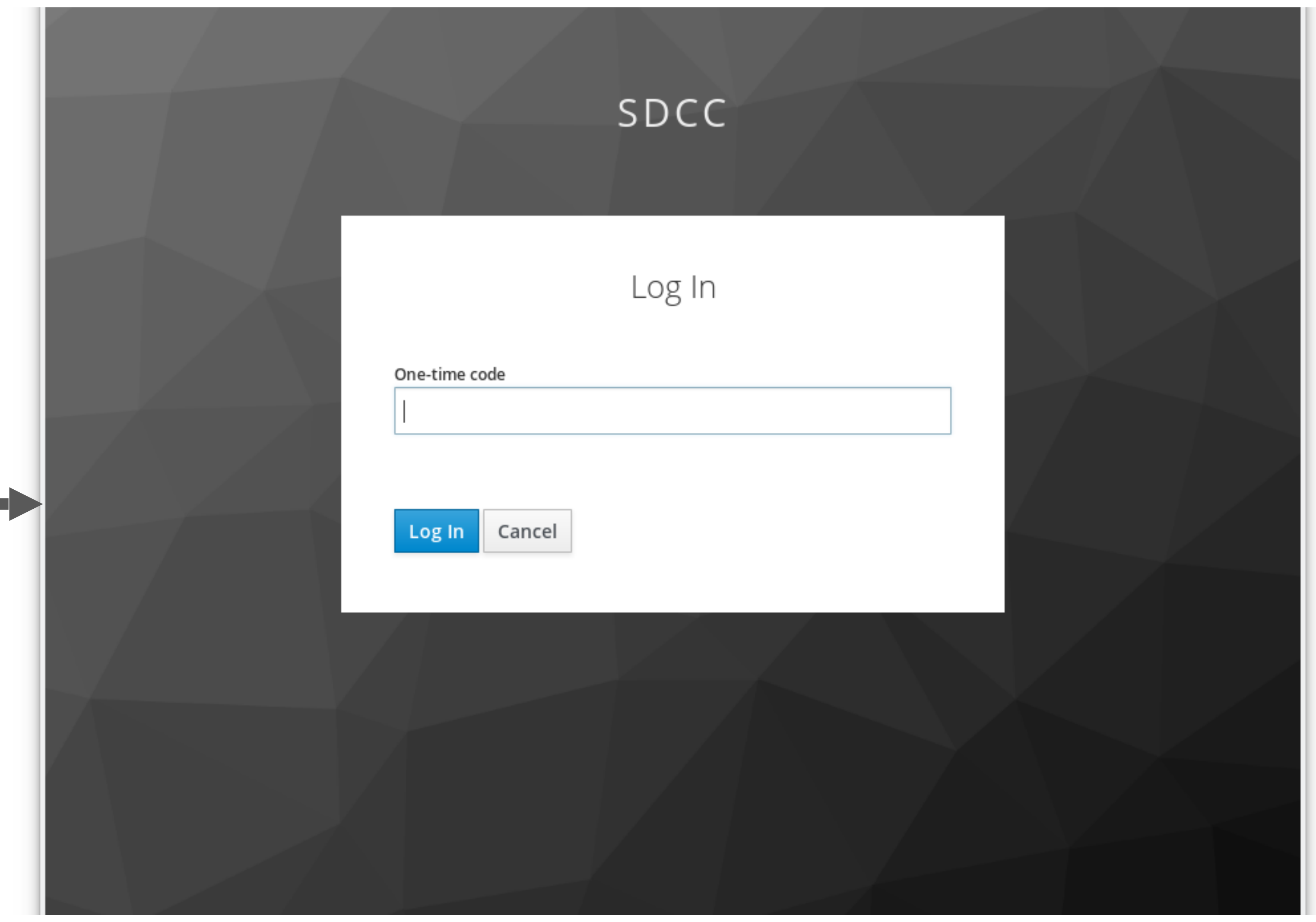
522944
Company VPN
jdoe@company.com

Evernote
john.doe

003128
Facebook
John Doe

GitHub
jdoe

252552
Google
john.doe1987@gmail.com



SDCC

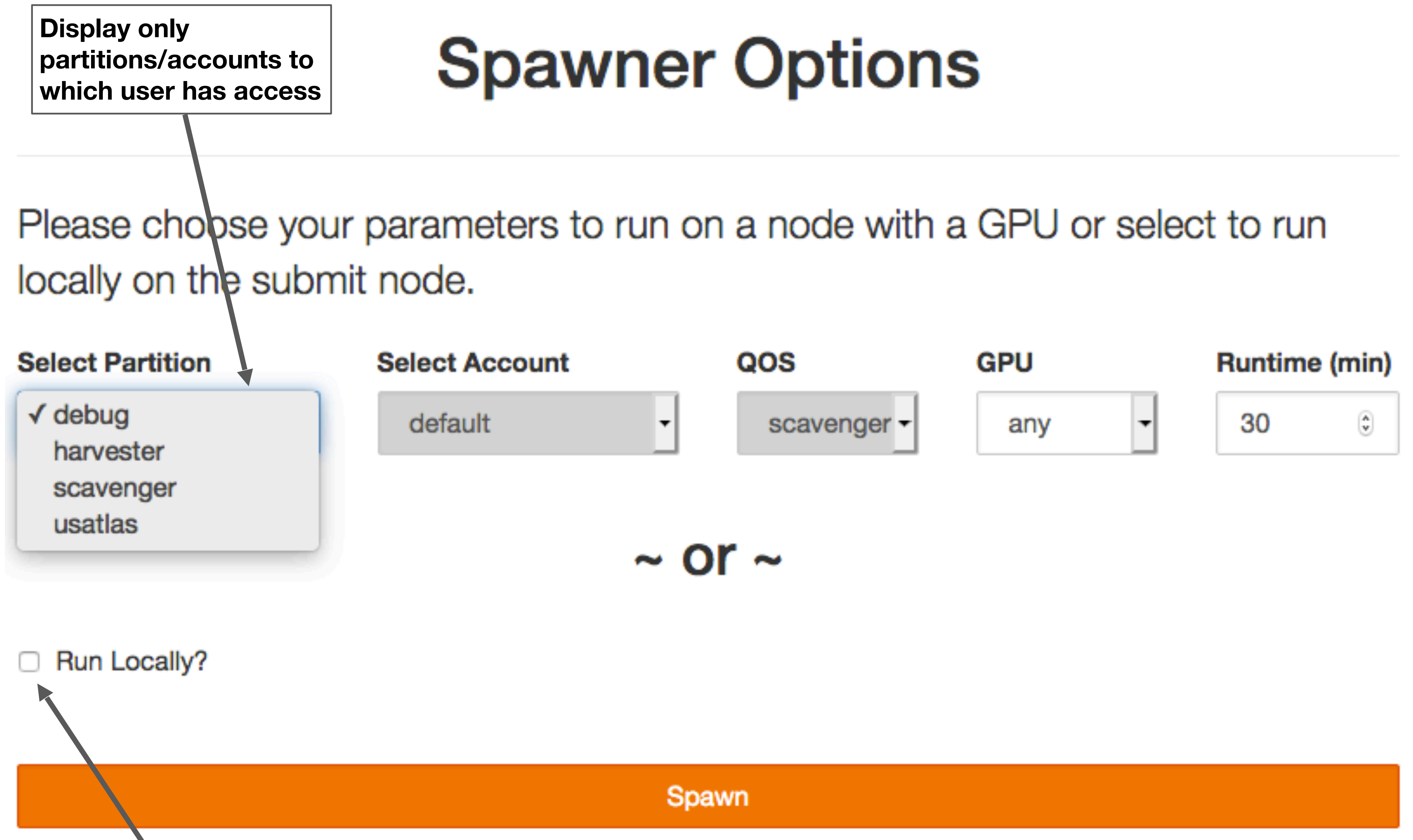
Log In

One-time code

Log In Cancel

Custom Slurm Spawner Interface

Display only partitions/accounts to which user has access



Spawner Options

Please choose your parameters to run on a node with a GPU or select to run locally on the submit node.

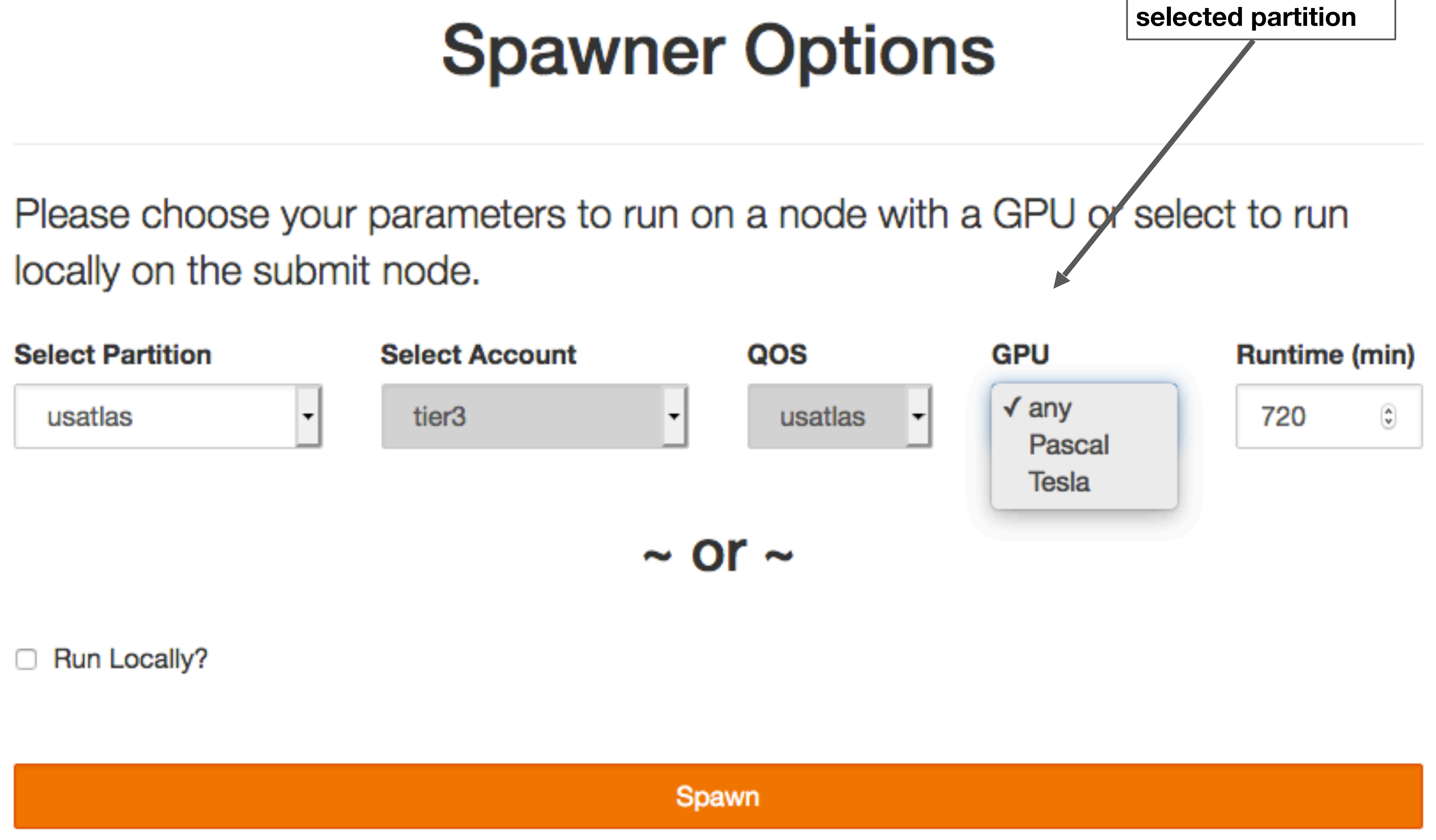
Select Partition	Select Account	QOS	GPU	Runtime (min)
✓ debug harvester scavenger usatlas	default	scavenger	any	30

~ or ~

☐ Run Locally?

Select here and will launch Local instead of Batch spawner

Account and Options defined by selected partition



Spawner Options

Please choose your parameters to run on a node with a GPU or select to run locally on the submit node.

Select Partition	Select Account	QOS	GPU	Runtime (min)
usatlas	tier3	usatlas	✓ any Pascal Tesla	720

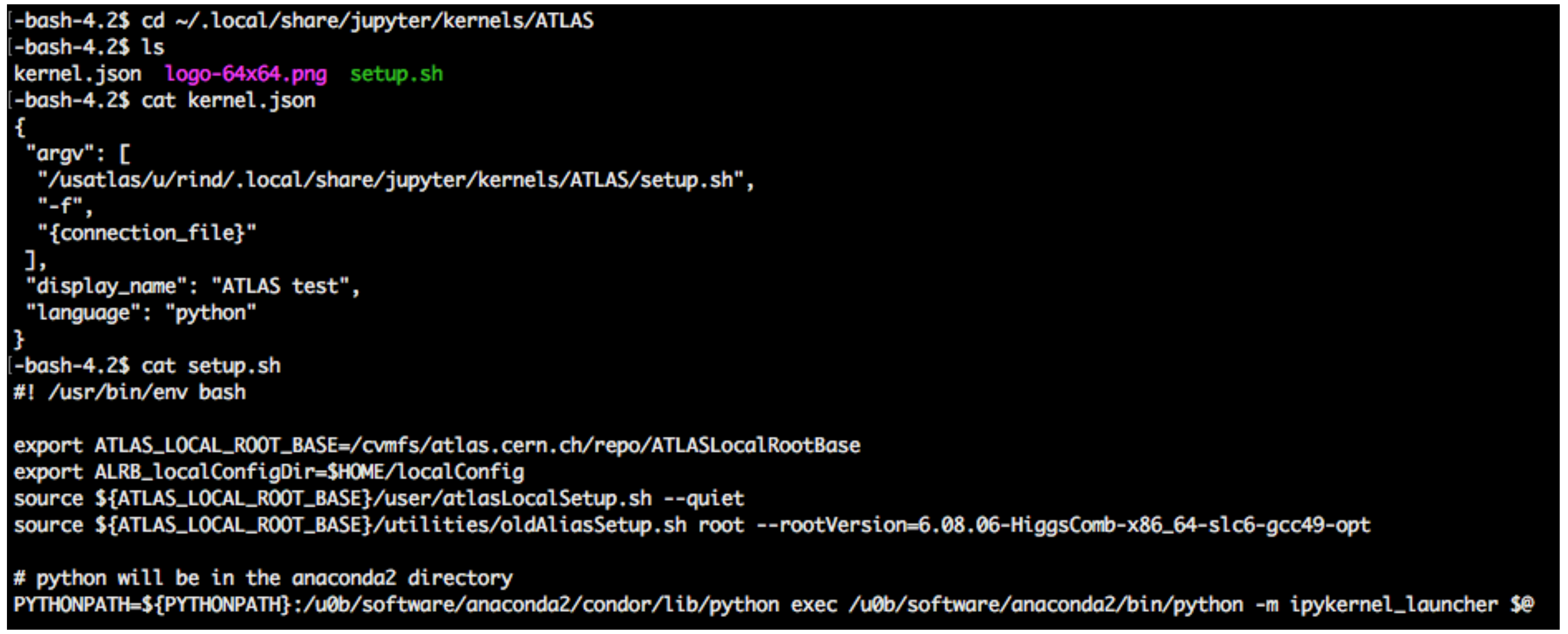
~ or ~

☐ Run Locally?

*** For form spawner code see https://github.com/fubarwrangler/sdcc_jupyter**

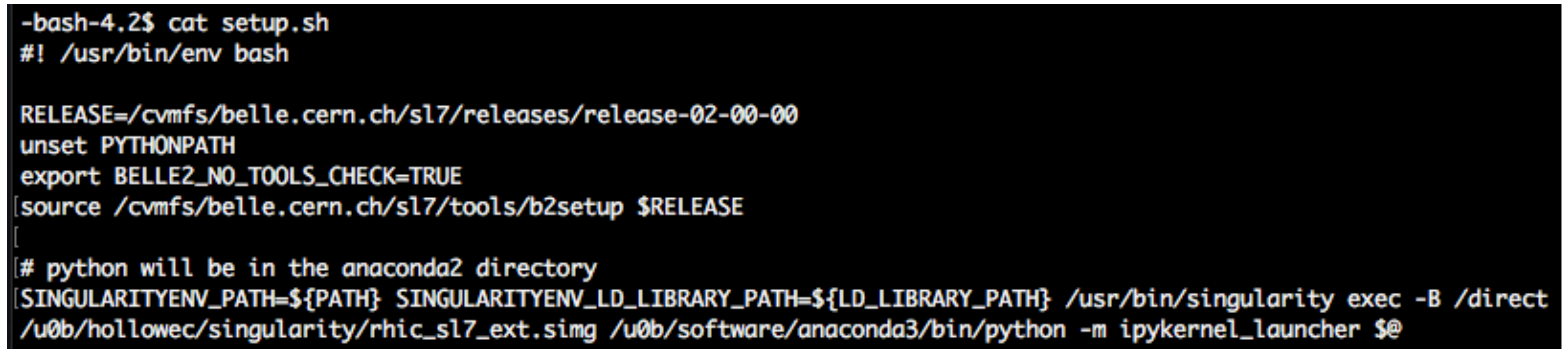
Challenges of Experiment Environments

- When you get a session (start a notebook-server), which environment?
  - Customization at the kernel level or via notebook-server container
- Whose problem is setting up the environments?
  - Work for a software librarian



```
-bash-4.2$ cd ~/.local/share/jupyter/kernels/ATLAS
-bash-4.2$ ls
kernel.json  logo-64x64.png  setup.sh
-bash-4.2$ cat kernel.json
{
 "argv": [
  "/usatlas/u/rind/.local/share/jupyter/kernels/ATLAS/setup.sh",
  "-f",
  "{connection_file}"
 ],
 "display_name": "ATLAS test",
 "language": "python"
}
-bash-4.2$ cat setup.sh
#! /usr/bin/env bash

export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export ALRB_localConfigDir=$HOME/localConfig
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh --quiet
source ${ATLAS_LOCAL_ROOT_BASE}/utilities/oldAliasSetup.sh root --rootVersion=6.08.06-HiggsComb-x86_64-slc6-gcc49-opt

# python will be in the anaconda2 directory
PYTHONPATH=${PYTHONPATH}:/u0b/software/anaconda2/condor/lib/python exec /u0b/software/anaconda2/bin/python -m ipykernel_launcher $@
```

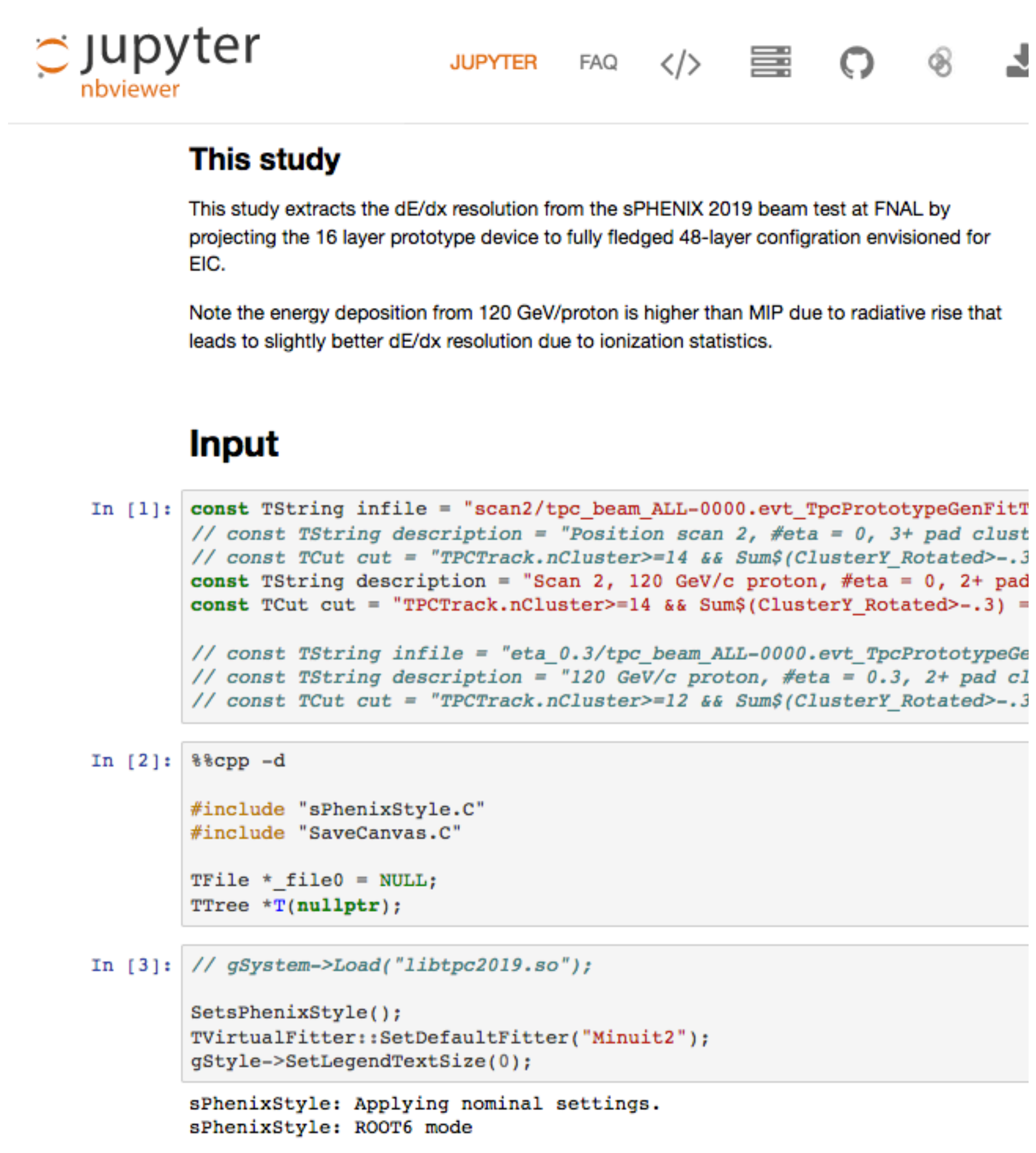
Kernel Customization



```
-bash-4.2$ cat setup.sh
#! /usr/bin/env bash

RELEASE=/cvmfs/belle.cern.ch/sl7/releases/release-02-00-00
unset PYTHONPATH
export BELLE2_NO_TOOLS_CHECK=TRUE
source /cvmfs/belle.cern.ch/sl7/tools/b2setup $RELEASE

# python will be in the anaconda2 directory
SINGULARITYENV_PATH=${PATH} SINGULARITYENV_LD_LIBRARY_PATH=${LD_LIBRARY_PATH} /usr/bin/singularity exec -B /direct
/u0b/hollowec/singularity/rhic_sl7_ext.simg /u0b/software/anaconda3/bin/python -m ipykernel_launcher $@
```

Custom Container

Example: sPHENIX Test Beam



jupyter nbviewer JUPYTER FAQ

This study

This study extracts the dE/dx resolution from the sPHENIX 2019 beam test at FNAL by projecting the 16 layer prototype device to fully fledged 48-layer configration envisioned for EIC.

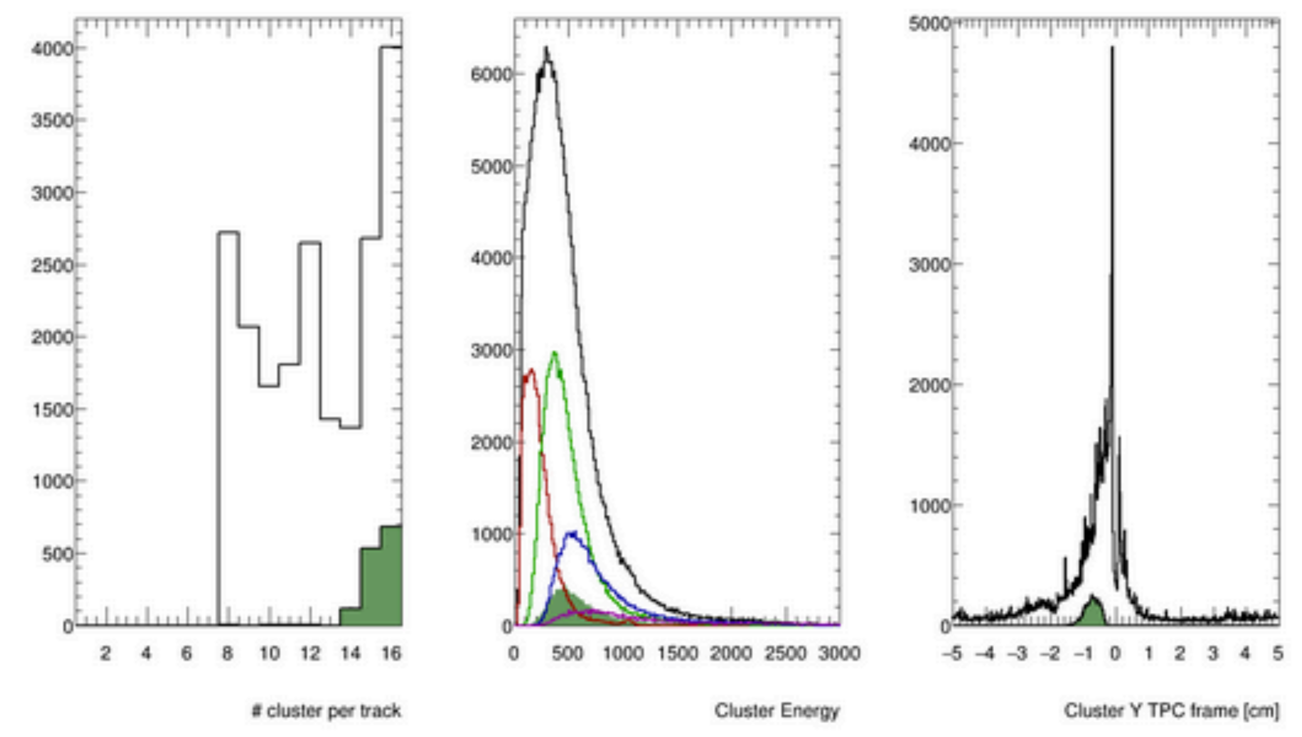
Note the energy deposition from 120 GeV/proton is higher than MIP due to radiative rise that leads to slightly better dE/dx resolution due to ionization statistics.

Input

```
In [1]: const TString infile = "scan2/tpc_beam_ALL-0000.evt_TpcPrototypeGenFitT
        // const TString description = "Position scan 2, #eta = 0, 3+ pad clust
        // const TCut cut = "TPCTrack.nCluster>=14 && Sum$(ClusterY_Rotated>-.3
        const TString description = "Scan 2, 120 GeV/c proton, #eta = 0, 2+ pad
        const TCut cut = "TPCTrack.nCluster>=14 && Sum$(ClusterY_Rotated>-.3) =

        // const TString infile = "eta_0.3/tpc_beam_ALL-0000.evt_TpcPrototypeGe
        // const TString description = "120 GeV/c proton, #eta = 0.3, 2+ pad cl
        // const TCut cut = "TPCTrack.nCluster>=12 && Sum$(ClusterY_Rotated>-.3
```

```
In [2]: %%cpp -d

        #include "sPhenixStyle.C"
        #include "SaveCanvas.C"

        TFile *_file0 = NULL;
        TTree *T(nullptr);
```

```
In [3]: // gSystem->Load("libtpc2019.so");

        SetsPhenixStyle();
        TVirtualFitter::SetDefaultFitter("Minuit2");
        gStyle->SetLegendTextSize(0);

        sPhenixStyle: Applying nominal settings.
        sPhenixStyle: ROOT6 mode
```



```
Info in <TCanvas::SaveSource>: C++ Macro file: scan2/tpc_beam_ALL-0000.
```

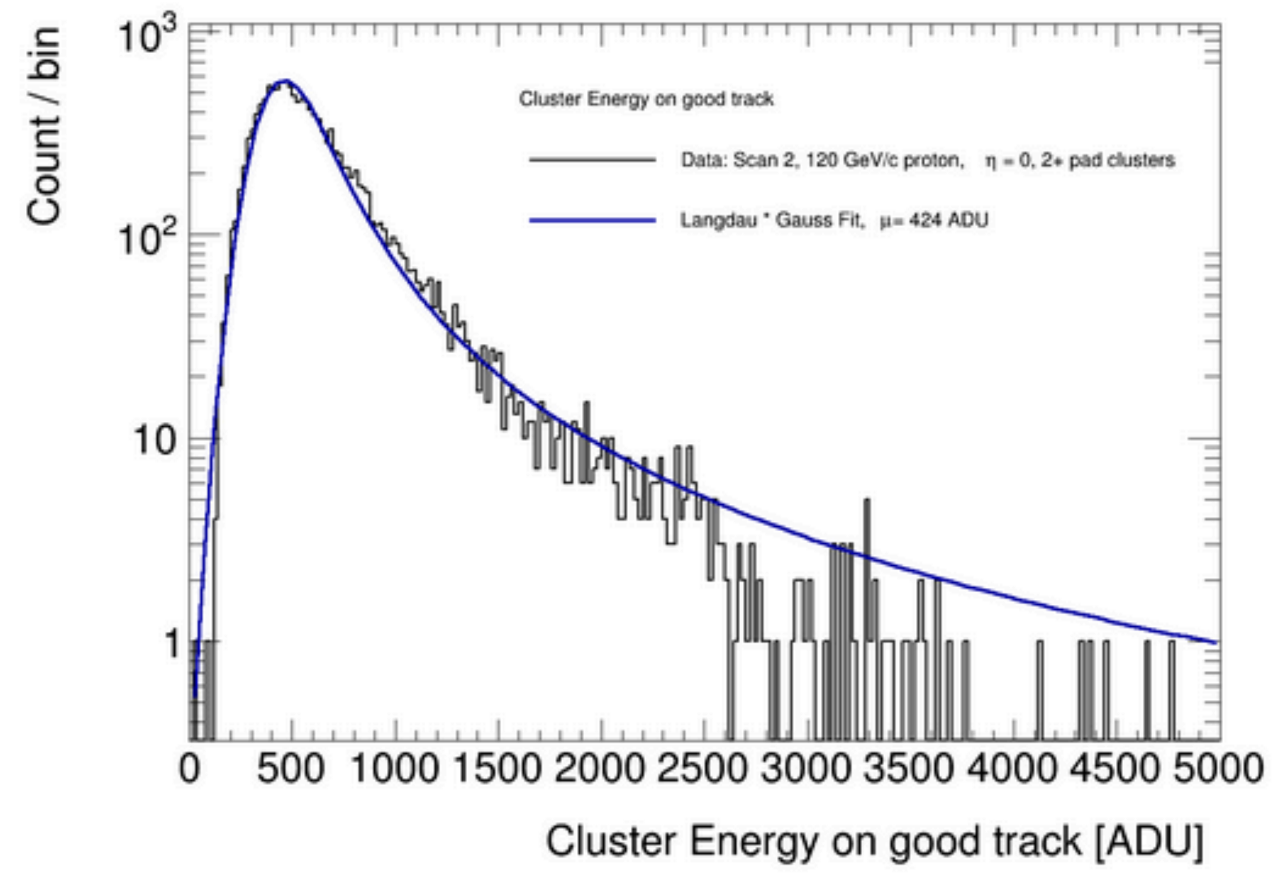
Note the cluster energy for 1-pad and 2-pad cluster (red, green) are significantly lower than that from 3/3+ pad clusters (blue, magenta). To be understood. For now, just analyzing 2+ pad clusters as the charge spread from zig-zag are expected to spread the charge to 3 pads.

PDF

```
In [7]: %%cpp -d


        Double_t langaufun(Double_t *x, Double_t *par)
        {
          //Fit parameters:
          //par[0]=Width (scale) parameter of Landau density
          //par[1]=Most Probable (MP, location) parameter of Landau density
          //par[2]=Total area (integral -inf to inf, normalization constant)
          //par[3]=Width (sigma) of convoluted Gaussian function
          //
          //In the Landau distribution (represented by the CERNLIB approximatio
          //the maximum is located at x=-0.22278298 with the location parameter
          //This shift is corrected within this function, so that the actual
          //maximum is identical to the MP parameter.
```

```
    hClusterEnergyFit->Draw("same");
    //    gPhiDistortion->Draw("p");

    TLegend *leg = new TLegend(.4, .7, .95, .9,  + "Cluster Energy on g
    leg->AddEntry(hClusterEnergy, TString("Data: ") + description, "l")
    leg->AddEntry(hClusterEnergyFit,
                Form("Langdau * Gauss Fit, #mu= %.0f ADU", hClusterEner
    leg->Draw();

    c1->Draw();
    SaveCanvas(c1,
             TString(_file0->GetName()) + TString(c1->GetName()), kFALS
}
```

```
****************************************
Minimizer is Minuit2
Chi2                      =      403.493
NDf                       =          164
Edm                       =  2.69546e-08
NCalls                    =          193
Width                     =      66.7173   +/-   1.25122        (limit
MP                        =      423.708   +/-   2.09786        (limit
Area                      =       276531   +/-   2386.94        (limit
GSigma                    =       89.174   +/-   2.32025        (limit
Info in <TCanvas::Print>: png file scan2/tpc_beam_ALL-0000.evt_TpcProto
```



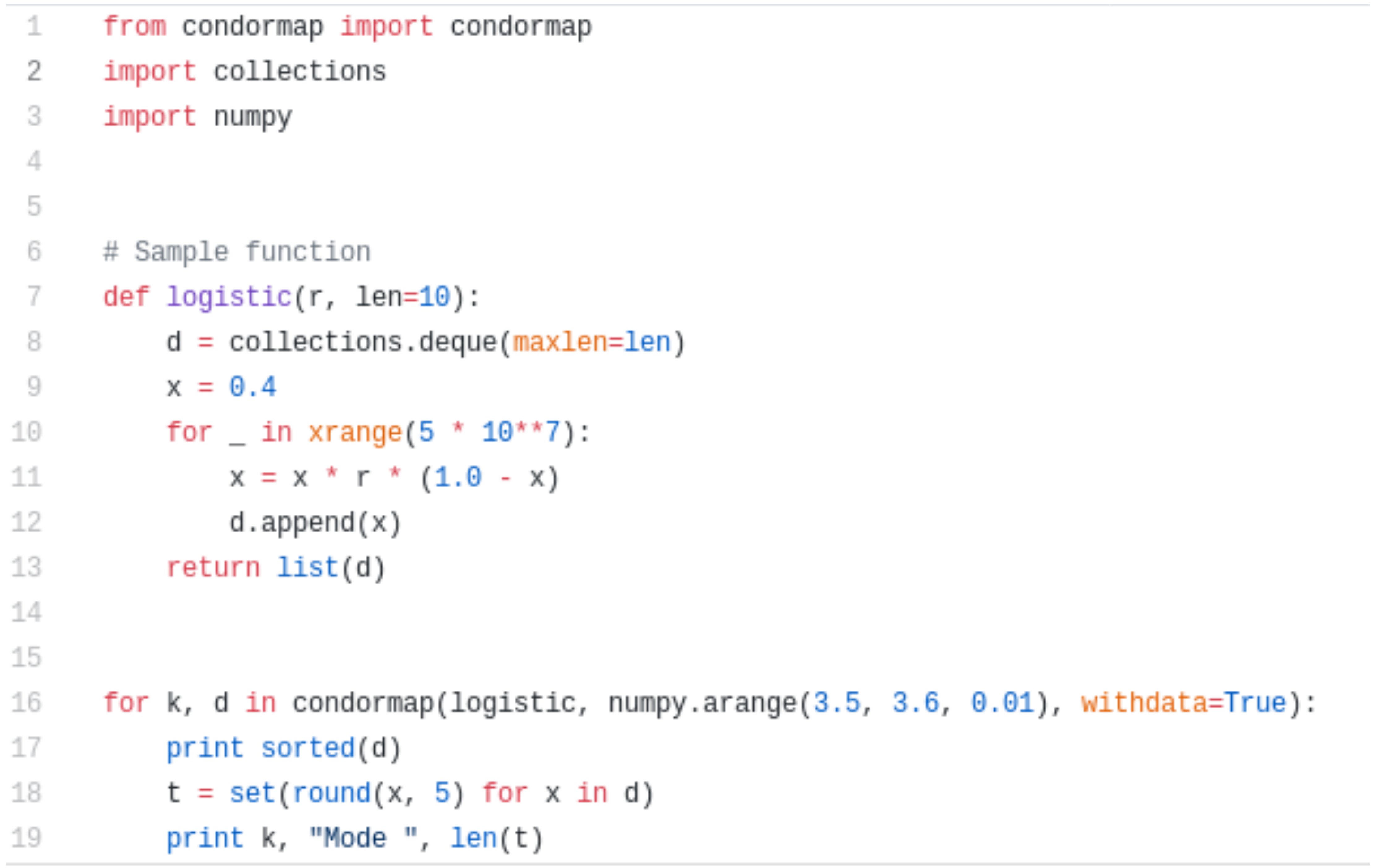
** Notebook analysis courtesy of Jin Huang using custom sPHENIX Root Kernel

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center

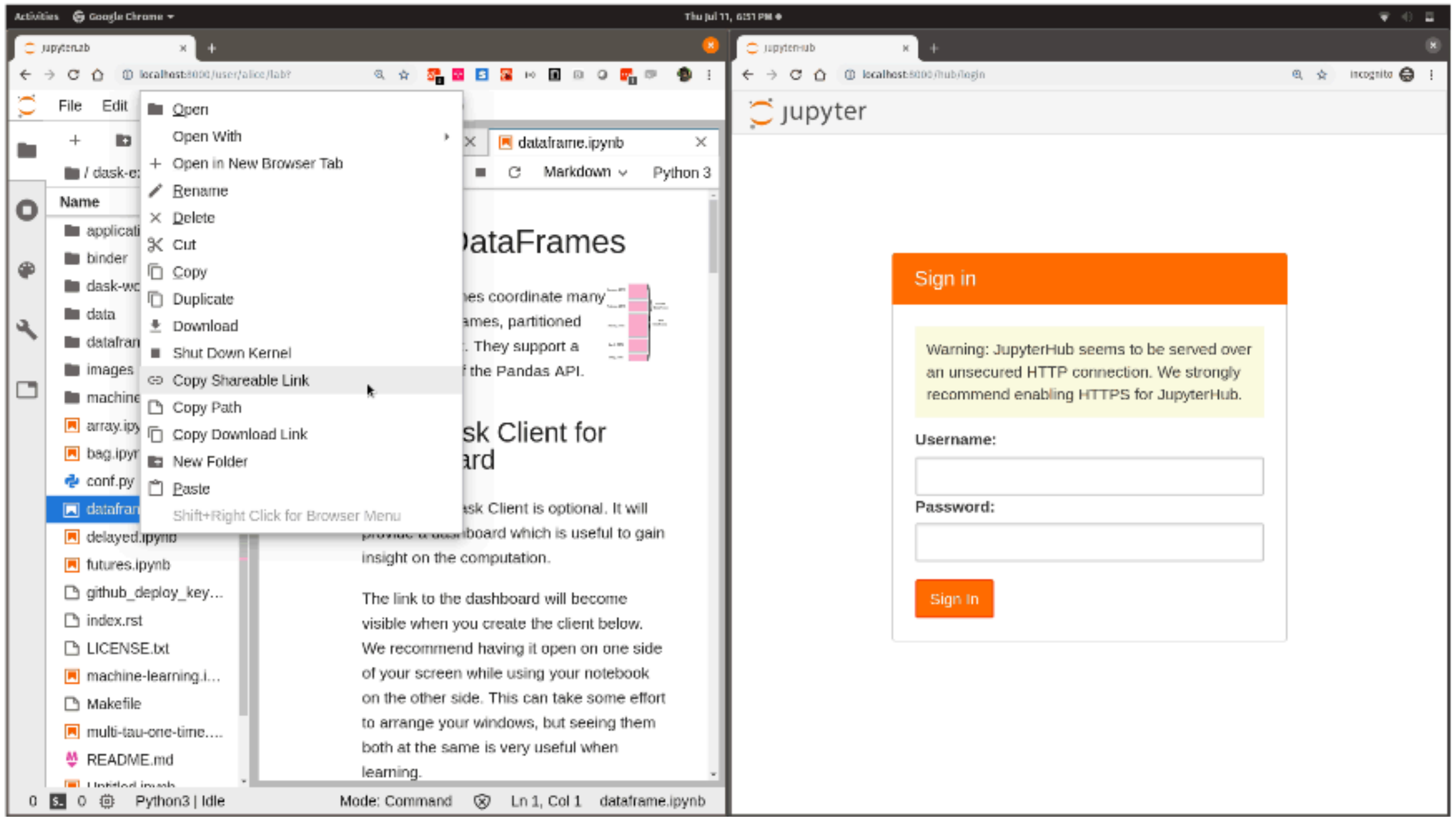
Orchestration: Integrating Jupyter with Compute

- How to make it easier to use compute from Jupyter?
  - HTMap library from condor
  - Dask / IPyParallel / Parsl etc...
- Goal: abstract away the fact that you are using a batch system at all
  - Either through trivial substitutes
    - map()→htmap()
  - Or through cell "magics"
    - %slurm or equivalent
  - Or via nice pythonic decorators that submit to batch systems (e.g. Dask-jobqueue)



```
from condormap import condormap
import collections
import numpy


# Sample function
def logistic(r, len=10):
    d = collections.deque(maxlen=len)
    x = 0.4
    for _ in xrange(5 * 10**7):
        x = x * r * (1.0 - x)
        d.append(x)
    return list(d)


for k, d in condormap(logistic, numpy.arange(3.5, 3.6, 0.01), withdata=True):
    print sorted(d)
    t = set(round(x, 5) for x in d)
    print k, "Mode ", len(t)
```

Notebook Sharing: Short Term



ataFrames

es coordinate many
ames, partitioned
. They support a
f the Pandas API.

sk Client for
ard

ask Client is optional. It will
provide a dashboard which is useful to gain
insight on the computation.

The link to the dashboard will become
visible when you create the client below.
We recommend having it open on one side
of your screen while using your notebook
on the other side. This can take some effort
to arrange your windows, but seeing them
both at the same is very useful when
learning.

Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

- Low-effort, short-term sharing between users on the same Hub
- Sender creates shareable link that provides last saved version of notebook to link recipient
  - Short-term link expires after certain time
  - Link encodes notebook options, such as container, to ensure compatible software environment
- See https://github.com/danielballan/jupyterhub-share-link

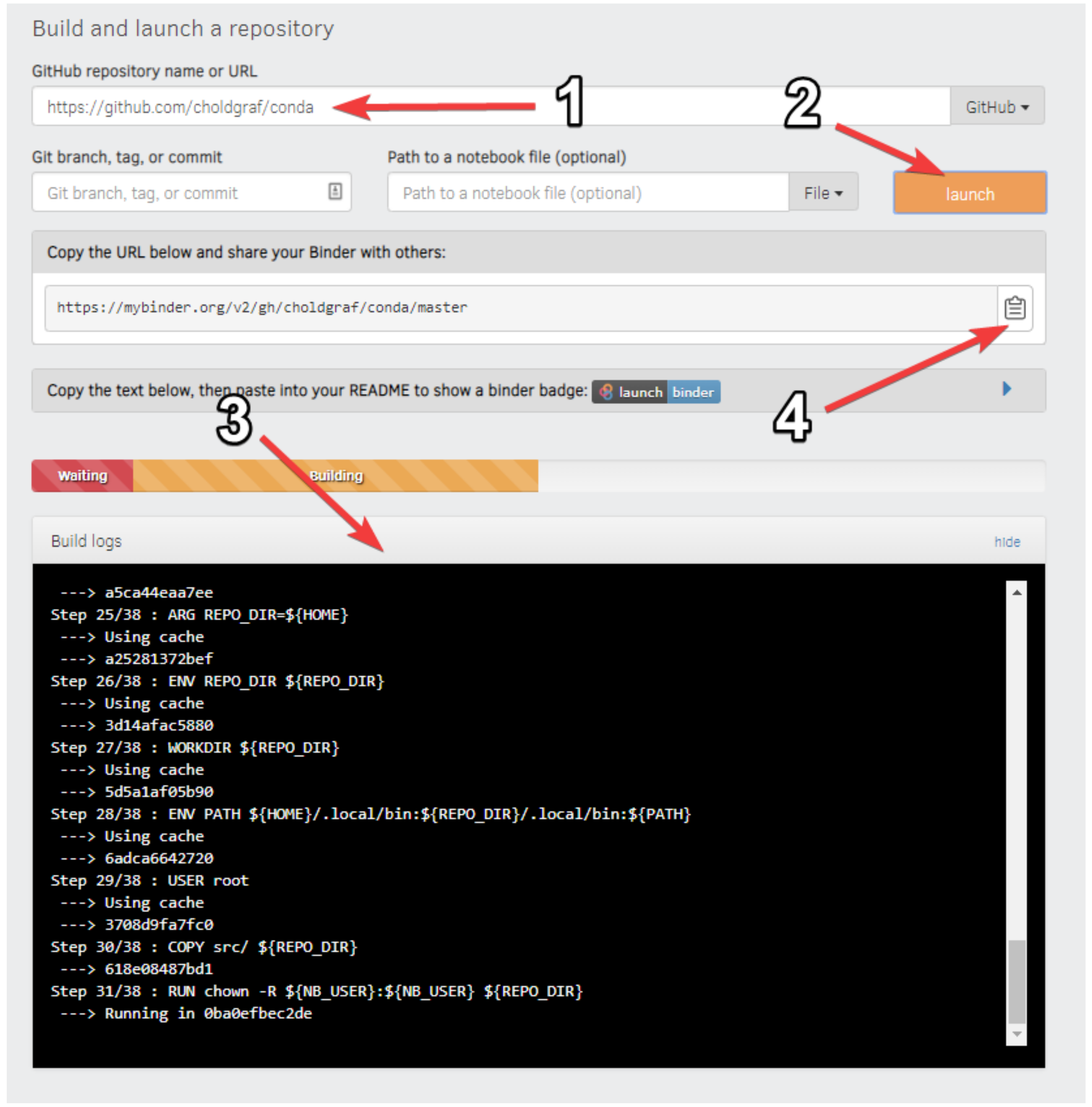
*** Courtesy Daniel Allan, illustrative gif: https://github.com/danielballan/jupyterhub-share-link/blob/master/demo.gif?raw=true**

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center

Notebook Archiving/Sharing

- Prepare a gallery of notebooks on a local Binder deployment, with a carefully defined software environment that anyone can recreate from a git repo with standard environment specs (e.g. requirements.txt)
    1. Enter URL of the repo
    2. Clicking "launch"
    3. Waiting and watching the build logs
    4. Copy a special link that will route directly to a Jupyter notebook running in a container that has repo contents and all software needed to run it successfully.
- Easy way for people to try your code and get running immediately
- Tightly coupled to Kubernetes and Docker, but developing similar workflows on HPC using Singularity


Build and launch a repository
GitHub repository name or URL
https://github.com/choldgraf/conda
1
2
GitHub
Git branch, tag, or commit
Git branch, tag, or commit
Path to a notebook file (optional)
Path to a notebook file (optional)
File
launch
Copy the URL below and share your Binder with others:
Copy the text below, then paste into your README to show a binder badge:
launch binder
3
4
Waiting
Building
Build logs
hide

```
https://mybinder.org/v2/gh/choldgraf/conda/master
```

```
 ---> a5ca44eaa7ee
Step 25/38 : ARG REPO_DIR=${HOME}
 ---> Using cache
 ---> a25281372bef
Step 26/38 : ENV REPO_DIR ${REPO_DIR}
 ---> Using cache
 ---> 3d14afac5880
Step 27/38 : WORKDIR ${REPO_DIR}
 ---> Using cache
 ---> 5d5a1af05b90
Step 28/38 : ENV PATH ${HOME}/.local/bin:${REPO_DIR}/.local/bin:${PATH}
 ---> Using cache
 ---> 6adca6642720
Step 29/38 : USER root
 ---> Using cache
 ---> 3708d9fa7fc0
Step 30/38 : COPY src/ ${REPO_DIR}
 ---> 618e08487bd1
Step 31/38 : RUN chown -R ${NB_USER}:${NB_USER} ${REPO_DIR}
 ---> Running in 0ba0efbec2de
```

*** Courtesy Daniel Allan**

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center

Conclusions

- The SDCC at BNL is deploying a Jupyterhub infrastructure enabling scientists from multiple disciplines to access our diverse HTC and HPC computing resources

- System designed to meet facility requirements with minimal impact on the backend

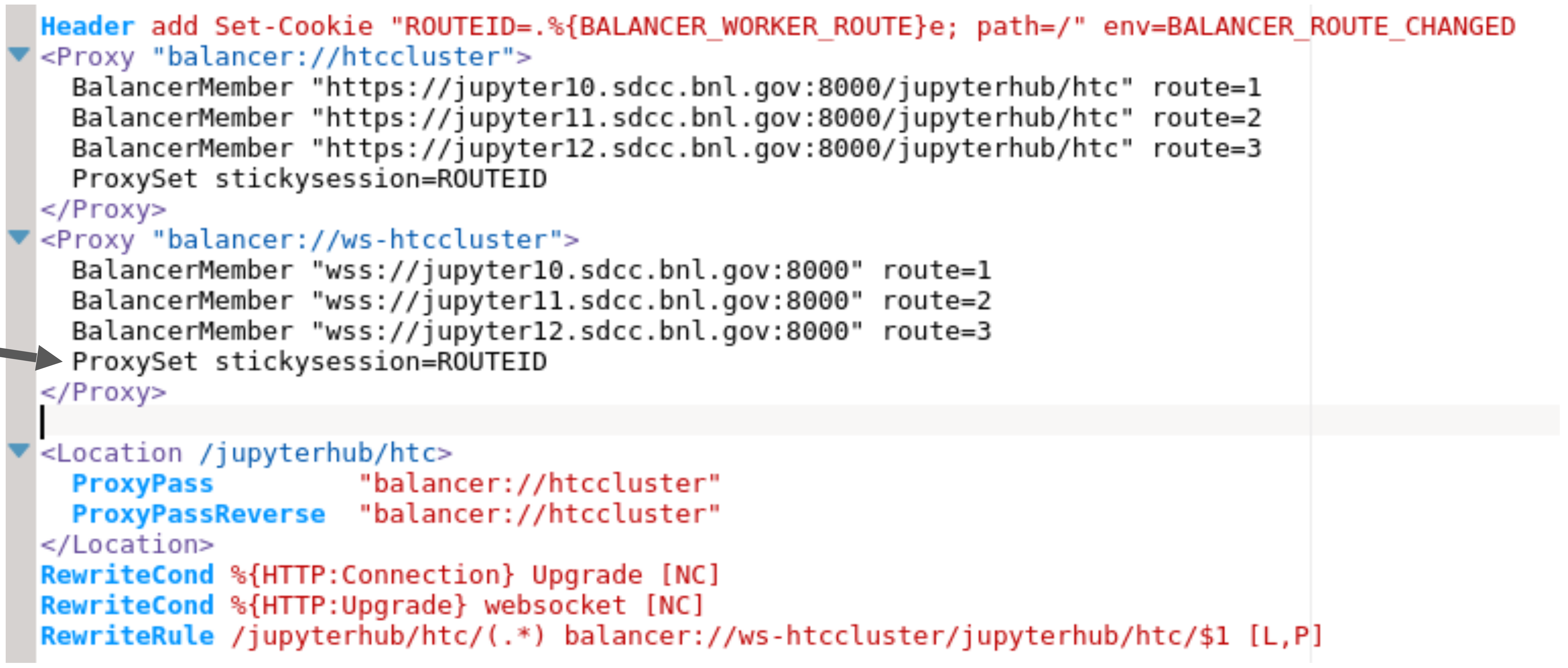
- Built-in support for experiment-based computing environment with a number of flexible access modes and workflows

- Continuing to develop new techniques for user collaboration

Extra Slides

HTTP Frontend Configuration

- Authentication via Mellon plugin (for Keycloak)
- Subdivide URL space for different hub servers
  - /jupyterhub/$cluster for HTC/HPC/others
- Load-balancing configuration
  - Need cookie for sticky-sessions
  - Newest apache on RHEL7
    - Requires websockets support



```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy "balancer://htccluster">
  BalancerMember "https://jupyter10.sdcc.bnl.gov:8000/jupyterhub/htc" route=1
  BalancerMember "https://jupyter11.sdcc.bnl.gov:8000/jupyterhub/htc" route=2
  BalancerMember "https://jupyter12.sdcc.bnl.gov:8000/jupyterhub/htc" route=3
  ProxySet stickysession=ROUTEID
</Proxy>
<Proxy "balancer://ws-htccluster">
  BalancerMember "wss://jupyter10.sdcc.bnl.gov:8000" route=1
  BalancerMember "wss://jupyter11.sdcc.bnl.gov:8000" route=2
  BalancerMember "wss://jupyter12.sdcc.bnl.gov:8000" route=3
  ProxySet stickysession=ROUTEID
</Proxy>

<Location /jupyterhub/htc>
  ProxyPass          "balancer://htccluster"
  ProxyPassReverse   "balancer://htccluster"
</Location>
RewriteCond %{HTTP:Connection} Upgrade [NC]
RewriteCond %{HTTP:Upgrade} websocket [NC]
RewriteRule /jupyterhub/htc/(.*) balancer://ws-htccluster/jupyterhub/htc/$1 [L,P]
```