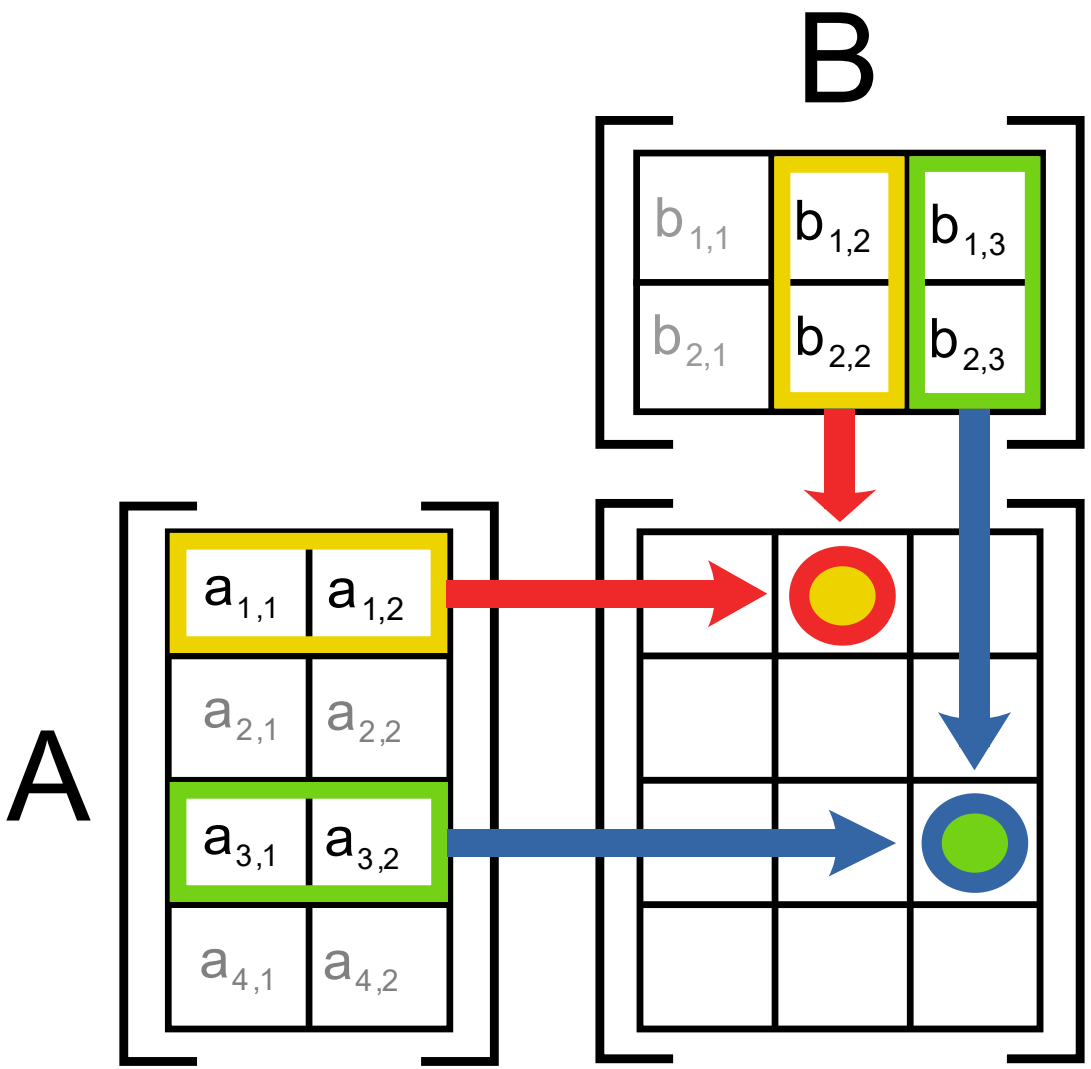
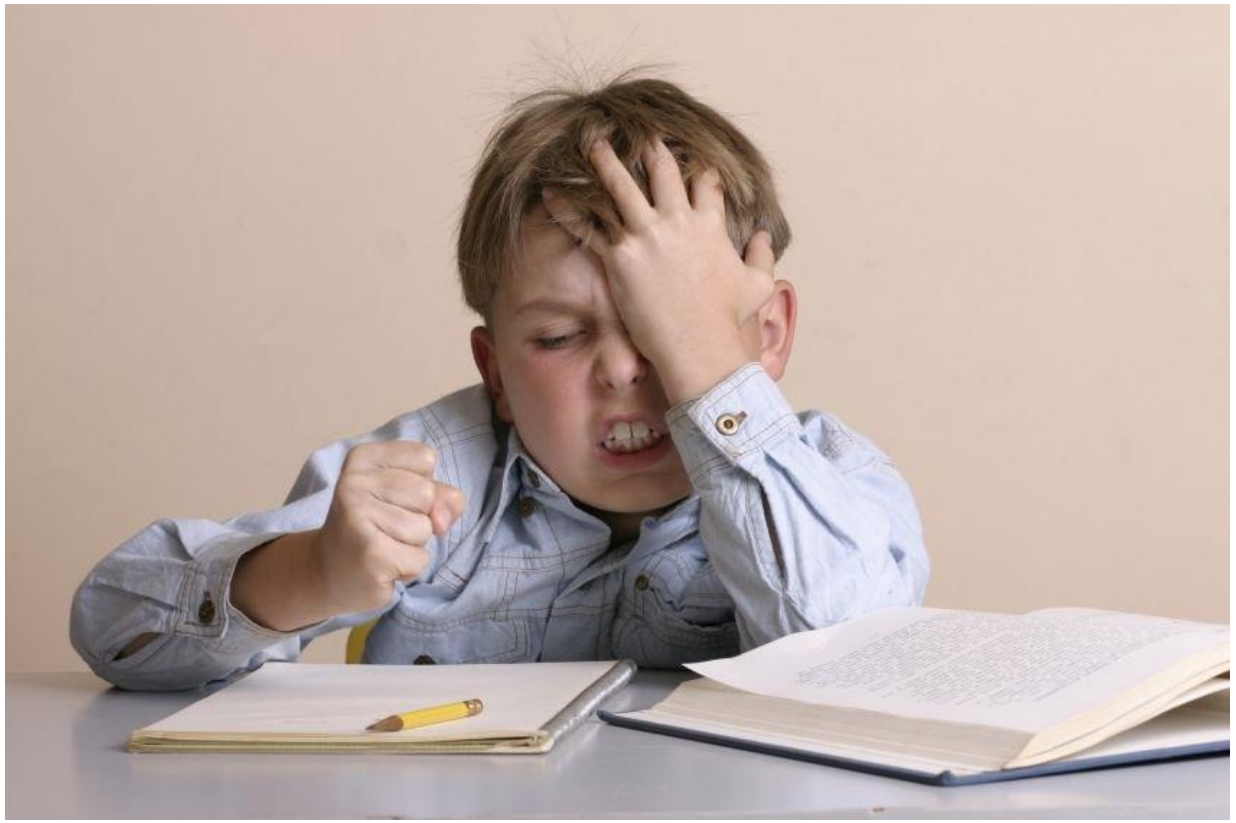
Python: Swiss-Army Glue

+

Josh Karpel <karpel@wisc.edu>
Graduate Student, Yavuz Group
UW-Madison Physics Department

My Research: Matrix Multiplication





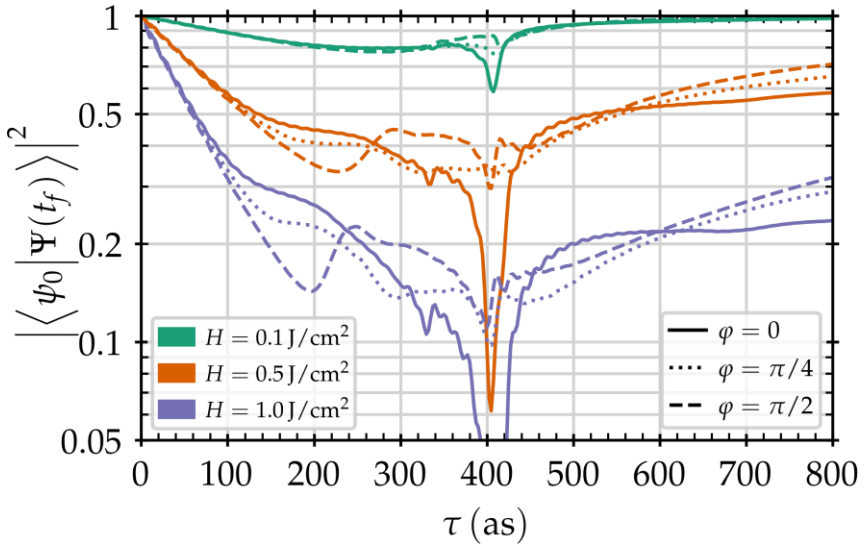
My Research: Computational Quantum Mechanics

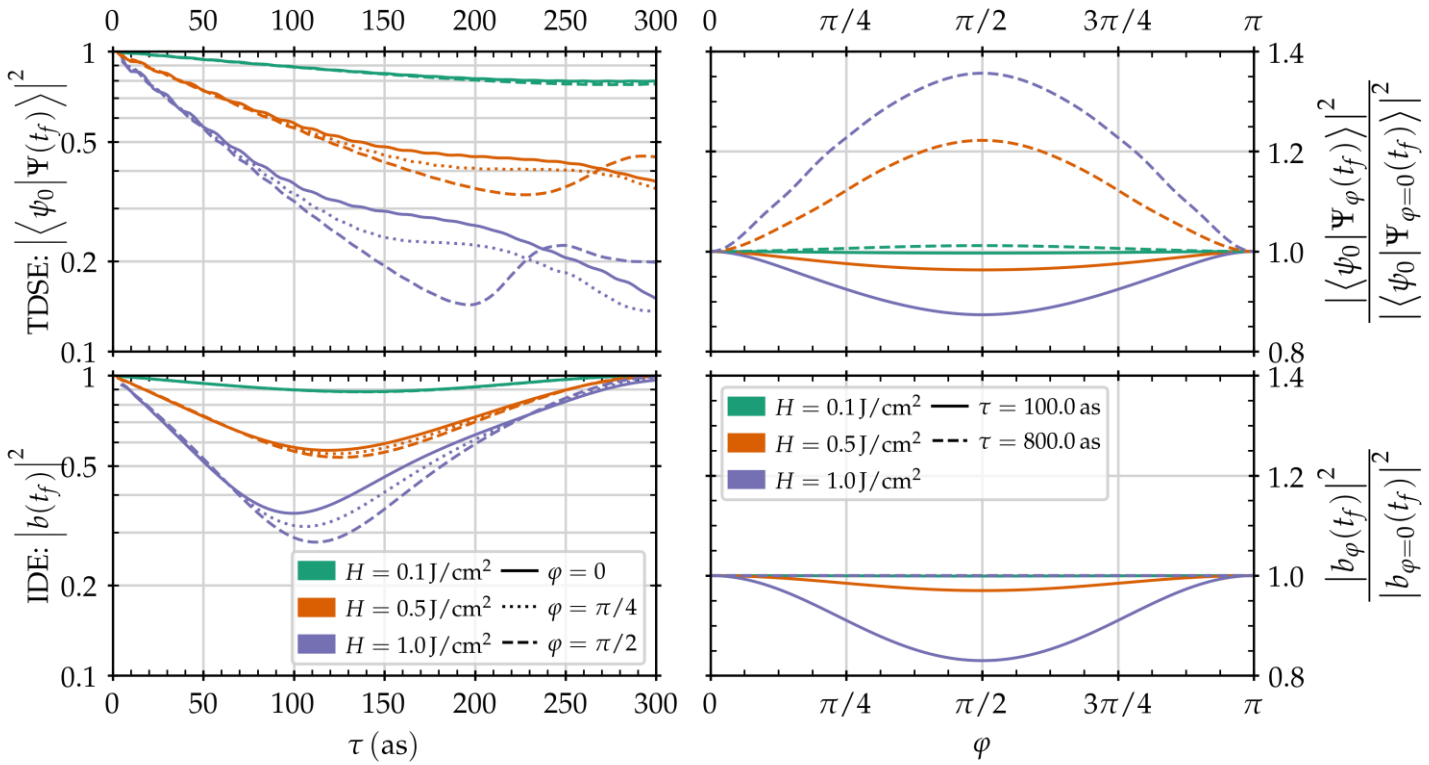
Why HTC?

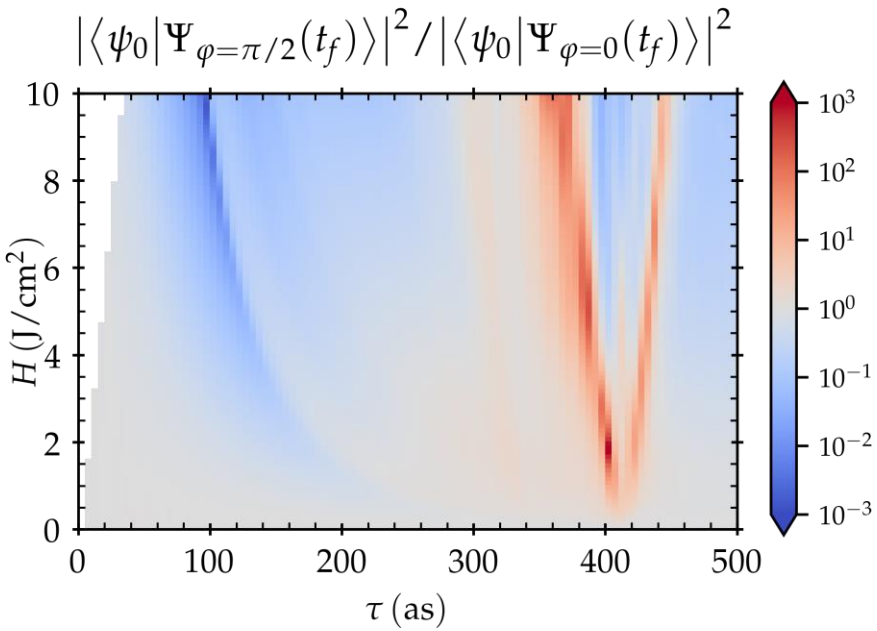
HUGE PARAMETER SCANS

How HTC?

Manage jobs w/o big infrastructure







https://doi.org/10.1364/OL.43.002583

Using Python for Cluster Tooling

Create

Create jobs programmatically: “questionnaires”

Run

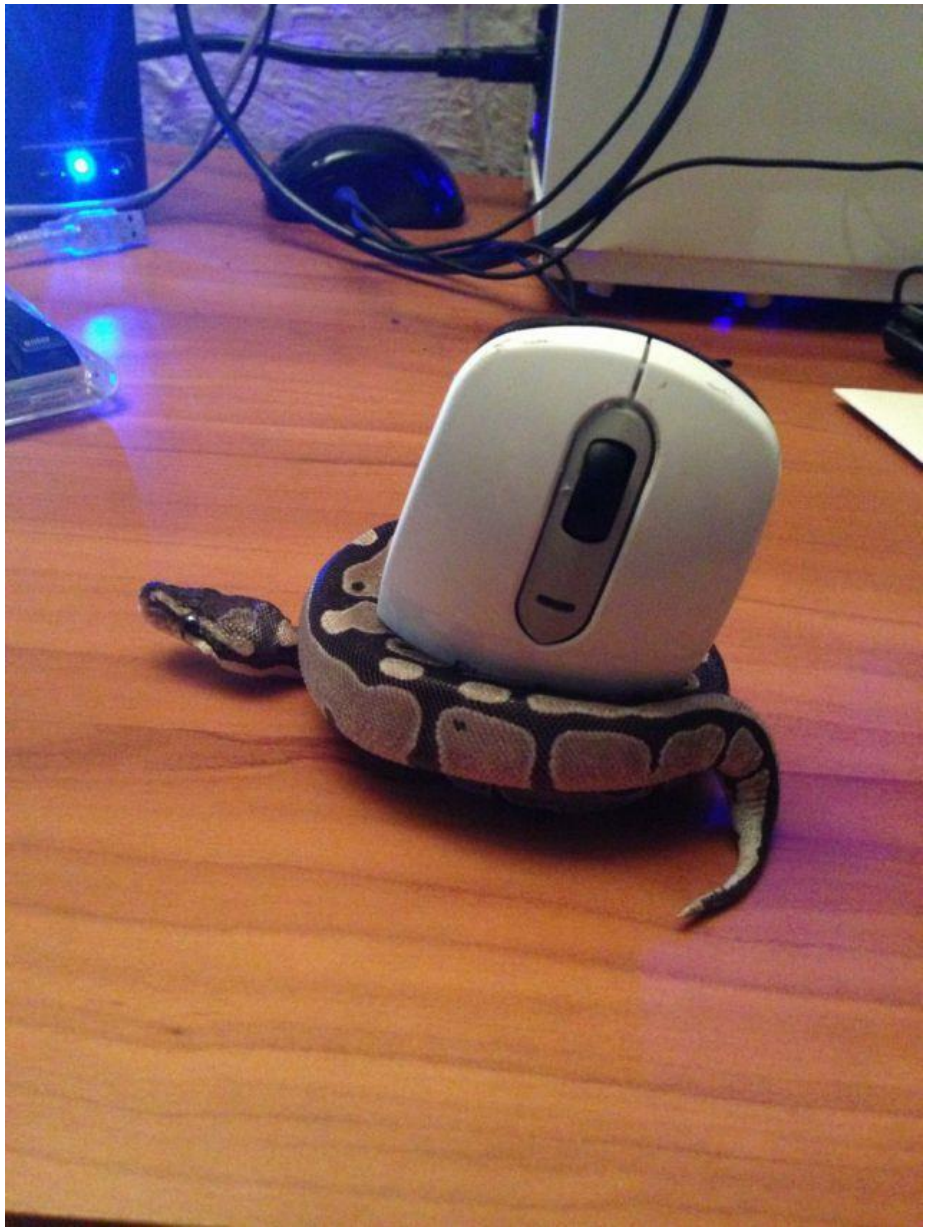
Computation:
`numpy`
`scipy`
`cython`
...

Store rich data: `pickle`

Analyze

Automate file transfer: `paramiko`

Processing:
`pandas`
`sqlite`
`matplotlib`
...



Using compiled C/Fortran code

```
>>> import numpy as np
>>> x = np.array(list(range(10)))
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> x * 2
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

>>> x ** 2
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])

>>> np.dot(x, x)
285
```

Also see: Cython, Numba, F2PY, etc.

Scientific Python stack is full-featured

Other Things People Like	Python Equivalents
Mathematica's symbolic mathematics	`sympy`
Mathematica Notebooks / MATLAB's command window	`IPython Notebooks`
MATLAB's multidimensional arrays	`numpy`
MATLAB's plotting tools	`matplotlib`
Pre-implemented numerical routines	`scipy`

Plus all the power of Python as a general-purpose language!

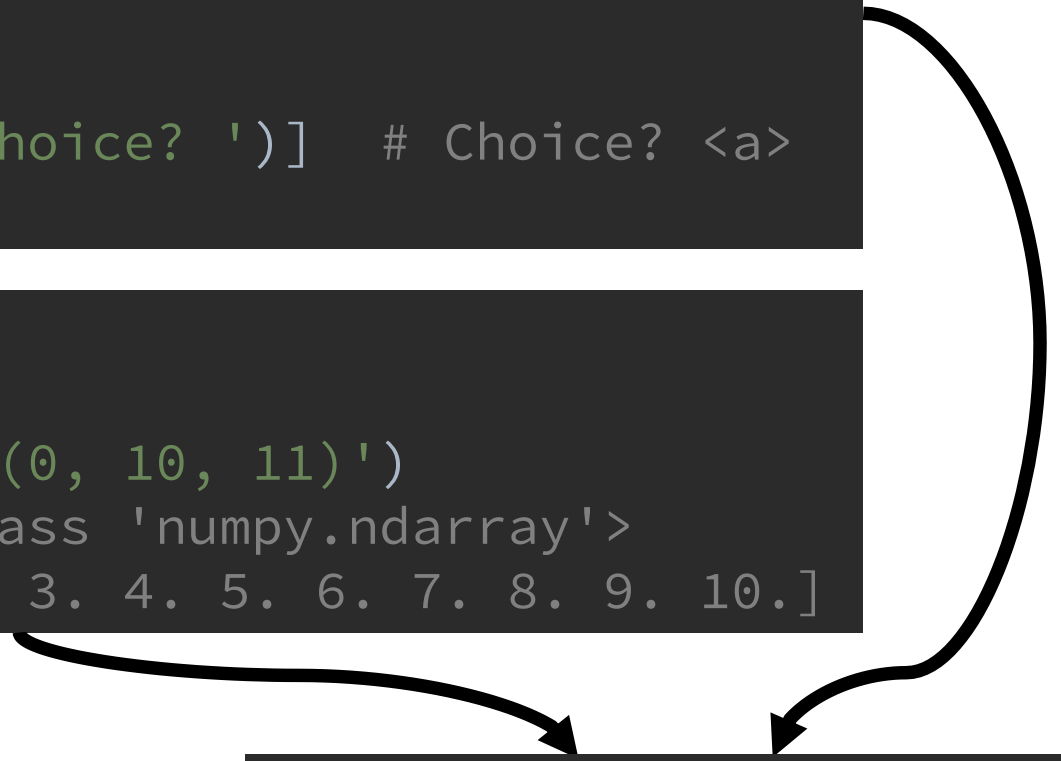
Generate jobs programmatically via “questionnaires”

```
$ ./create_job__tdse_scan.py demo --dry
Mesh Type [cyl | sph | harm] [Default: harm] >
R Bound (Bohr radii) [Default: 200] > 100
R Points per Bohr Radii [Default: 10] > 20
l points [Default: 500] >
[WARNING] ~ Predicted memory usage per Simulation is >15.3 MB
Mask Inner Radius (in Bohr radii)? [Default: 80.0] >
Mask Outer Radius (in Bohr radii)? [Default: 100.0] >
<MORE QUESTIONS>
Generated 75 Specifications
Job batch name? [Default: demo] >
Flock and Glide? [Default: y] > n
Memory (in GB)? [Default: 1] > .8
Disk (in GB)? [Default: 5] > 3
Creating job directory and subdirectories...
Saving Specifications...
Writing Specification info to file...
Writing submit file...
```

Use `input` and `eval` (carefully!)

```
choices = {
    'a': 'hello',
    'b': 'goodbye',
}

choice = choices[input('Choice? ')]  # Choice? <a>
print(choice)  # hello
```

```
import numpy as np

array = eval('np.linspace(0, 10, 11)')
print(type(array))  # <class 'numpy.ndarray'>
print(array)  # [0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]
```



```
array = eval(input('Enter your array!'))
```

Generate jobs programmatically via questionnaires

```
[09:18 PM | karpel@submit-5 | ~/jobs/demo] $ ls -lh
total 236K
-rw-rw-r-- 1 karpel karpel  257 Apr  9 21:09 info.pkl
drwxrwxr-x 2 karpel karpel 4.0K Apr  9 21:09 inputs
drwxrwxr-x 2 karpel karpel 4.0K Apr  9 21:09 logs
drwxrwxr-x 2 karpel karpel 4.0K Apr  9 21:09 outputs
-rw-rw-r-- 1 karpel karpel  22K Apr  9 21:09 parameters.txt
-rw-rw-r-- 1 karpel karpel 186K Apr  9 21:09 specifications.txt
-rw-rw-r-- 1 karpel karpel  972 Apr  9 21:09 submit_job.sub
```

Advantages

- Avoids copy-paste issues
- Provide feedback during job creation to catch errors early
- Flexible enough to define new “types” of jobs without writing entirely new scripts
- Easy to generate metadata about job

Store rich data using `pickle`

```
import pickle

class Greeting:
    def __init__(self, words):
        self.words = words

    def yell(self):
        print(self.words.upper())

greeting = Greeting('hi!')

with open('foo.pkl', mode = 'wb') as file:
    pickle.dump(greeting, file)

with open('foo.pkl', mode = 'rb') as file:
    from_file = pickle.load(file)

print(from_file.words)  # hi!
from_file.yell()  # HI!
```

Advantages

- Works straight out of the box
- Avoid transforming to/from other data formats (CSV, JSON, HDF5, etc.)
- Implement self-checkpointing jobs easily

Gotchas

- Certain types of objects can’t be serialized
- Not as compressed as dedicated formats
- Can accidently break backwards-compatibility

Automate file transfer using paramiko

```
import paramiko

remote_host = 'submit-5.chtc.wisc.edu'
username = 'karpel'
key_path = 'wouldnt/you/like/to/know'

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.connect(remote_host,
            username = username,
            key_filename = key_path)
ftp = ssh.open_sftp()

ssh.exec_command('ls -l')  # returns stdin, stdout, stderr

ftp.put('local/path', 'my/big/fat/input/data')
ftp.get('path/to/completed/simulation', 'local/path')
```

Advantages

- Runs on a schedule
- Easy to control which files get downloaded
- Can hook directly into data processing

Gotchas

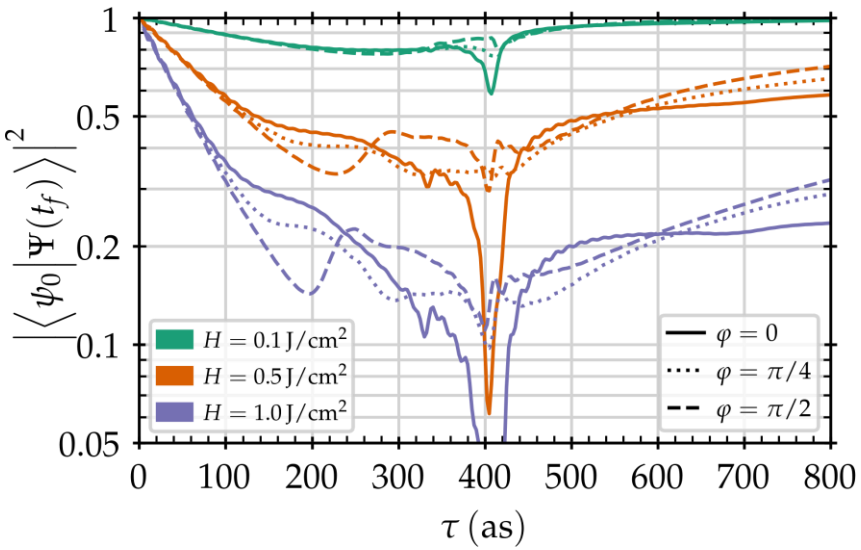
- Slow
- Occasional strange interactions with Dropbox/Box/Google Drive?

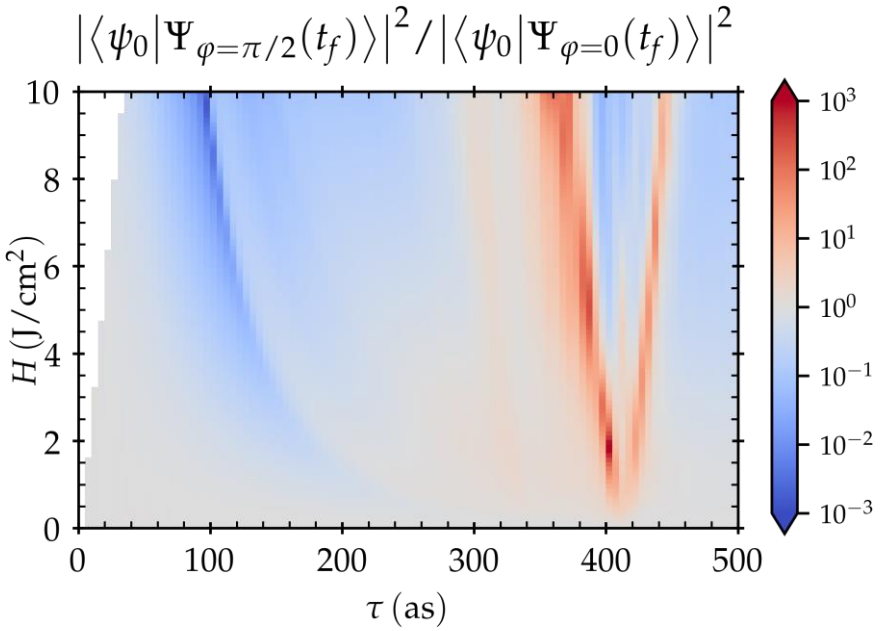
Process data using pandas

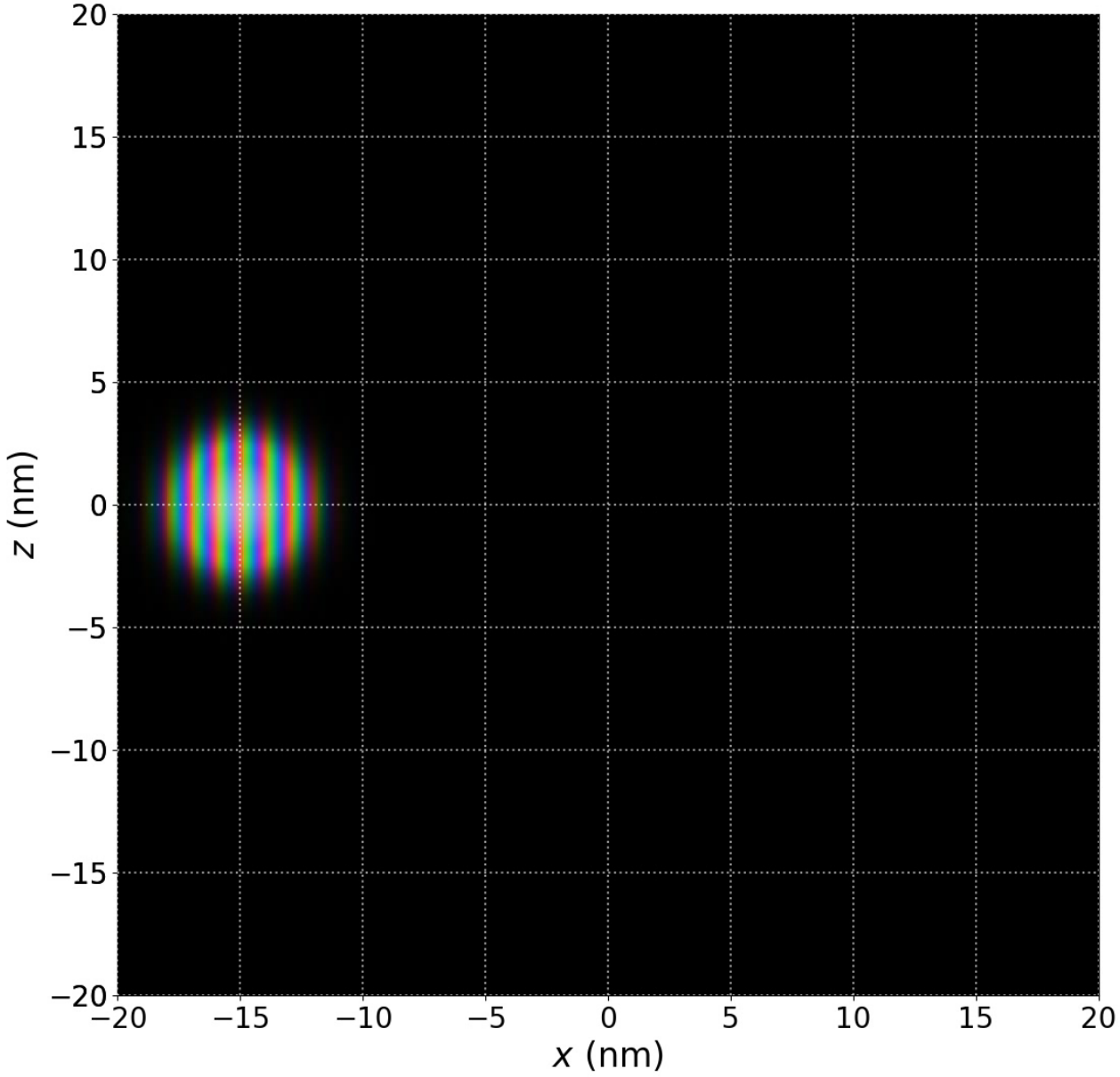
```python
import numpy as np
import pandas as pd

dates = pd.date_range('2018-01-01', periods = 6)
df = pd.DataFrame(
    np.random.randn(6, 4),
    index = dates,
    columns = list('ABCD'),
)

print(df)

######################

                   A         B         C         D
2018-01-01 -0.165014  0.721058  1.113825  1.778694
2018-01-02  1.774170  0.130640  1.089180 -0.812315
2018-01-03  1.167511  0.121111 -0.766156  1.816411
2018-01-04  0.103793  0.438878 -0.040532  0.238539
2018-01-05 -0.492766  1.466809 -0.384373  2.209309
2018-01-06 -1.304448  0.593538  0.055233  1.930035
```

```python
df = pd.read_excel(...)
df = pd.read_csv(...)
df = pd.read_hdf(...)
df = pd.read_json(...)
df = pd.read_pickle(...)
df = pd.read_sql(...)

df.to_excel(...)
df.to_csv(...)
df.to_hdf(...)
df.to_json(...)
df.to_pickle(...)
df.to_sql(...)

df.to_html(...)

# and more!
```

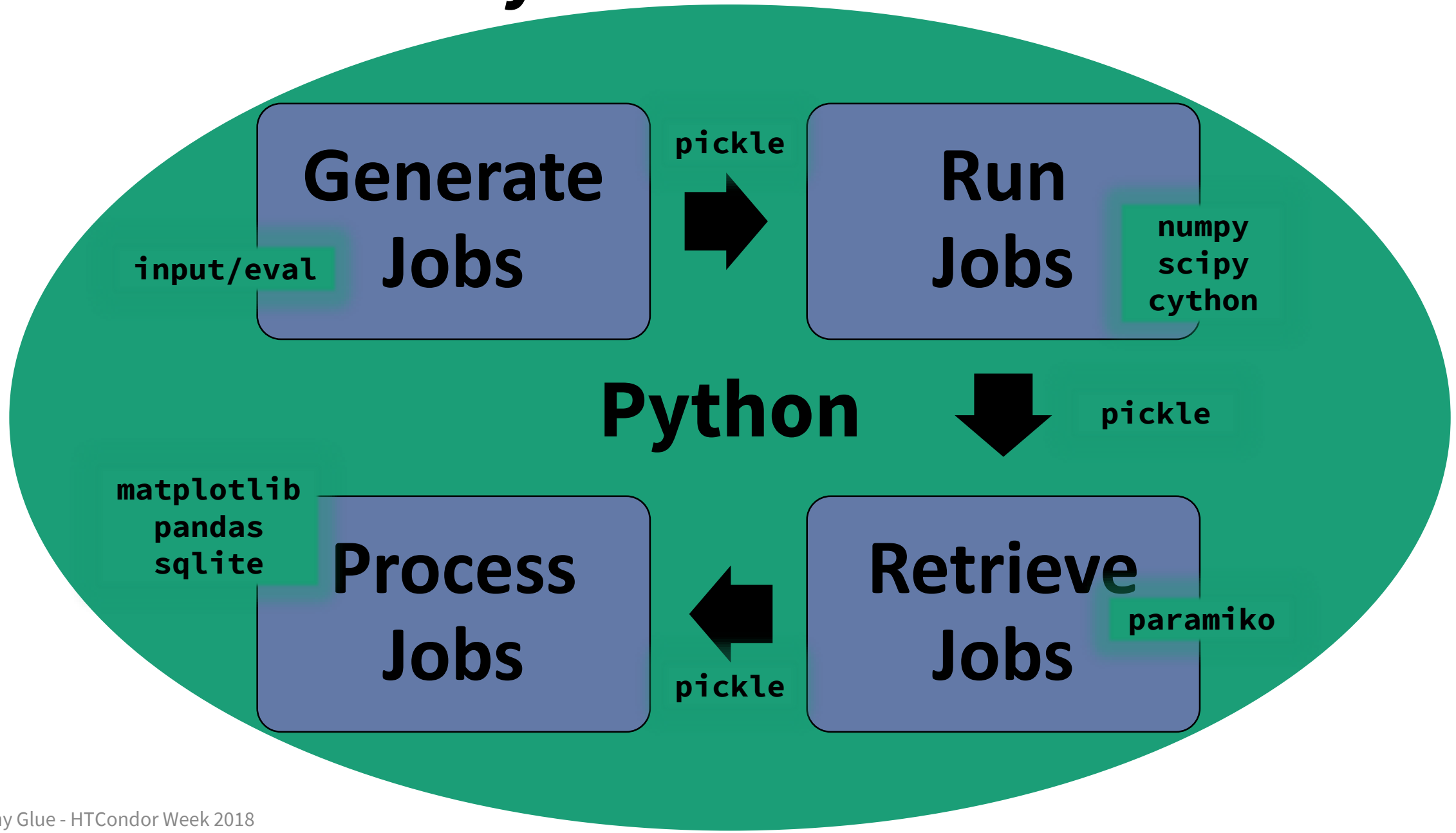
Visualize data using `matplotlib`







Python is **Swiss-Army Glue**



Where to go from here?

- My (extremely unstable) framework, Simulacra
- The HTCondor Python Bindings
- James Bourbeau's PyCondor

Your own ideas!