

# Parsl: Developing Interactive Parallel Workflows in Python using Parsl

Kyle Chard ([chard@uchicago.edu](mailto:chard@uchicago.edu))

Yadu Babuji, Anna Woodard, Zhuozhao Li, Ben Clifford,  
Ian Foster, Dan Katz, Mike Wilde, Justin Wozniak

<http://parsl-project.org>

# Parsl: Interactive parallel scripting in Python

Annotate functions to make Parsl  
*apps*

- Python apps call Python functions
- Bash apps call external applications

Apps return “futures”: a proxy for a result that might not yet be available

Apps run concurrently respecting data dependencies.

Natural parallel programming!

Parsl scripts are independent of where they run. Write once run anywhere!

```
pip install parsl
```

```
@python_app
def hello ():
    return 'Hello World!'

print(hello().result())
```



Hello World!

```
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
    print(f.read())
```



Hello World!

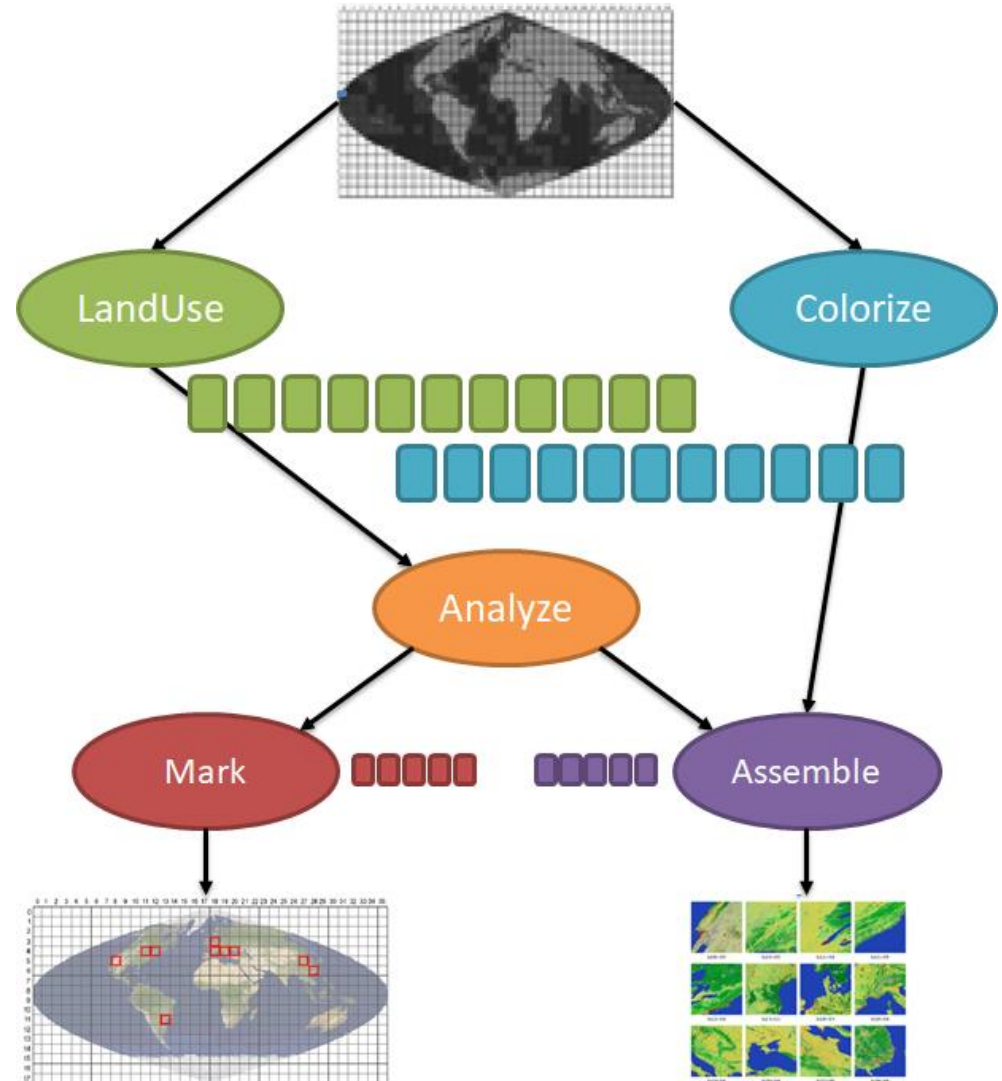
# Creating dynamic dataflows

Parsl creates a dynamic graph of tasks and their data dependencies

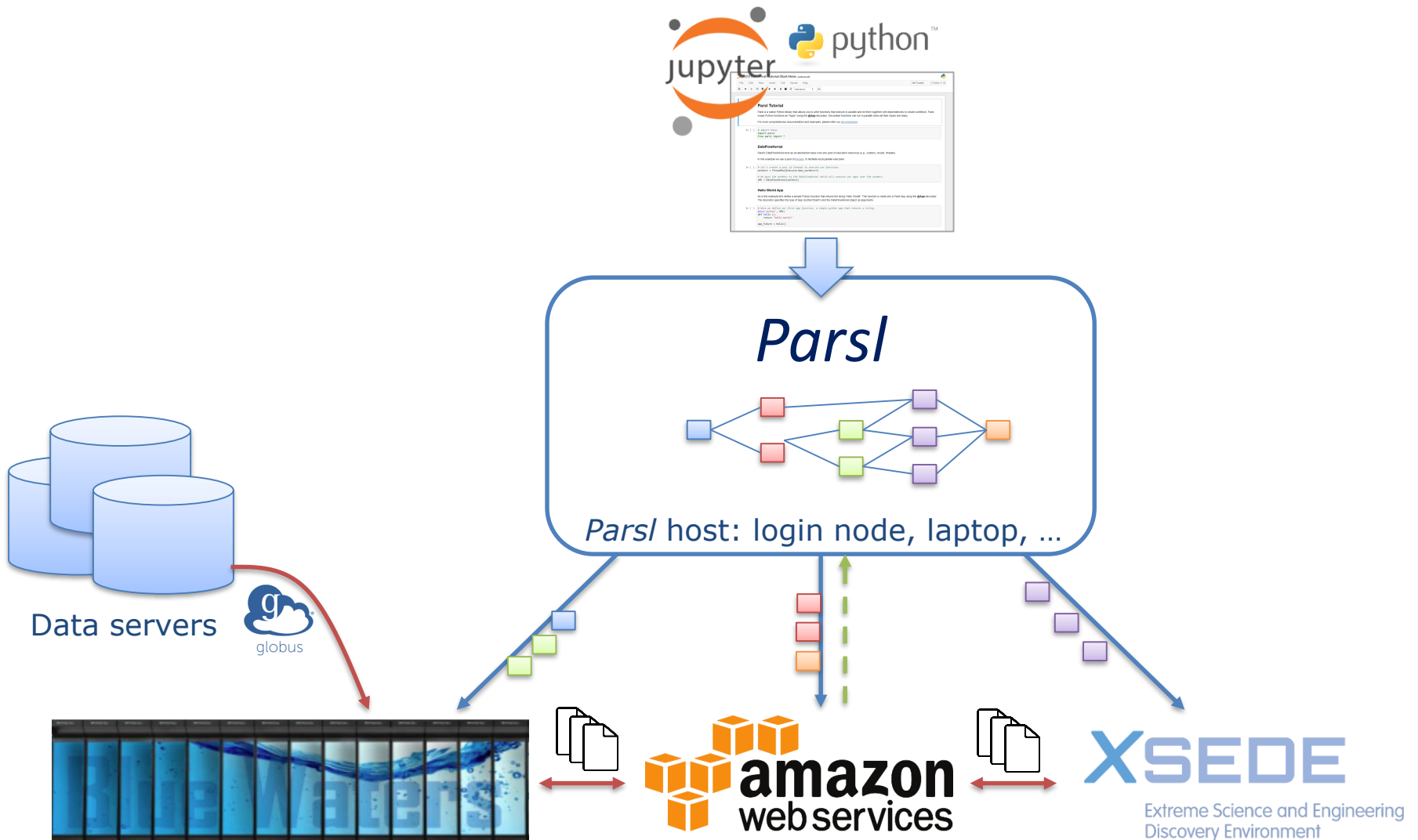
- Implicit based on shared input/output between apps

Tasks are only executed when their dependencies are met

Tasks without shared dependencies execute concurrently



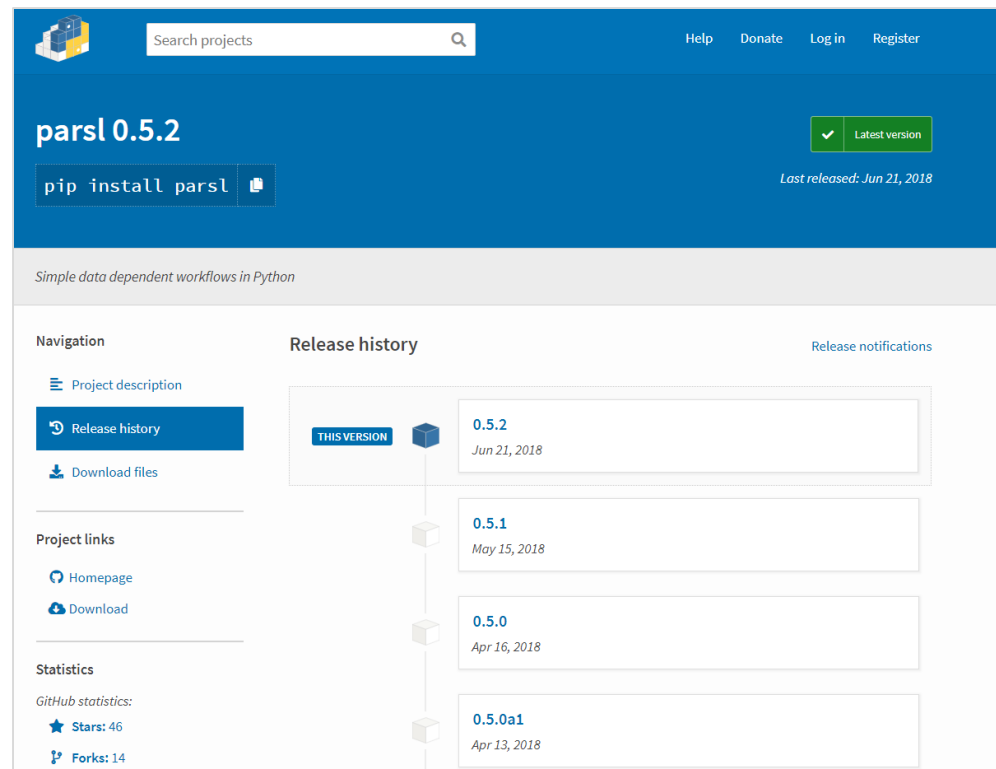
# Parsl in action: dynamic dataflow execution



# Parsl is Python

```
pip3 install parsl
```

- Use Python libraries natively
- Stage Python data transparently
- Integrates with Python ecosystem



The screenshot shows the PyPI page for the Parsl project. At the top, there is a search bar and navigation links for Help, Donate, Log in, and Register. The main header displays the project name 'parsl 0.5.2' and a green badge indicating it is the 'Latest version'. Below this, a button shows the command 'pip install parsl'. The page description is 'Simple data dependent workflows in Python'. The left sidebar contains navigation options: Project description, Release history (selected), and Download files. Below this are Project links for Homepage and Download, and Statistics showing 46 Stars and 14 Forks on GitHub. The main content area features a 'Release history' section with a vertical timeline of versions: 0.5.2 (Jun 21, 2018), 0.5.1 (May 15, 2018), 0.5.0 (Apr 16, 2018), and 0.5.0a1 (Apr 13, 2018). The 0.5.2 version is highlighted as 'THIS VERSION'.

# Parsl scripts are execution provider and execution model independent

The same script can be run locally, on grids, clouds, or supercomputers

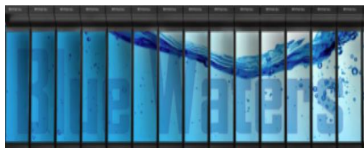
A single script may use many execution providers

- Local, Cloud (AWS, Azure, private), Slurm, Torque, Condor, Cobalt

A single script may use various execution models

- Threads, pilot jobs, extreme scale

Configuration file describes how to use resources



# Separation of code and execution environment

```
from libsubmit.channels import SSHChannel
from libsubmit.providers import SlurmProvider

import parsl
from parsl.config import Config
from parsl.executors.ipp import IPyParallelExecutor
from parsl.executors.threads import ThreadPoolExecutor
```

```
config = Config(
    executors=[
        IPyParallelExecutor(
            label='midway',
            provider=SlurmProvider(
                'westmere',
                channel=SSHChannel(
                    hostname='swift.rcc.uchicago.edu',
                    username='annawoodard'
                ),
                max_blocks=1000,
                nodes_per_block=1,
                tasks_per_node=6,
                overrides='module load singularity; module load Anaconda3/5.1.0; source activate parsl_py36'
            ),
        ),
        ThreadPoolExecutor(label='local', max_threads=2)
    ],
)

parsl.load(config)
```

```
@python_app(executors=['midway'])
def midway():
    return 'I run on midway'

@bash_app(executors=['local'])
def local():
    return 'echo "I run locally"'
```

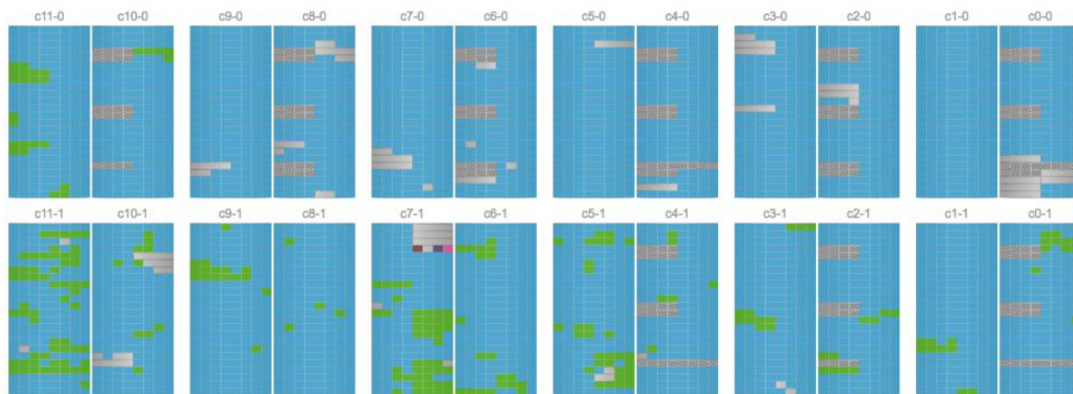
Pilot jobs on  
a cluster

Local threads

\* Config format for Parsl 0.6

# Extreme scale execution on Theta

4K Nodes (256K cores) for 72 hours in Singularity containers



256 Nodes 4000 Nodes 133  
105 c0-0c1s10n1

Total Running Jobs: 5

Job Id	Project	Nodes	Start Time	Run Time	Walltime	Queue	Mode
274219	LSSTADSP_DESC	4000	10:31:47 PM	00:02:19	23:59:00	default	script
274776	HHPMT_4	256	6:34:50 PM	03:59:15	06:00:00	default	script
275468	LQCD_VeloC	1	10:24:38 PM	00:09:27	01:00:00	debug-cache-quad	script
275453	OF_ICING	1	9:53:46 PM	00:40:19	01:00:00	debug-cache-quad	script
275467	Intel	1	10:19:44 PM	00:14:22	00:30:00	debug-cache-quad	interactive

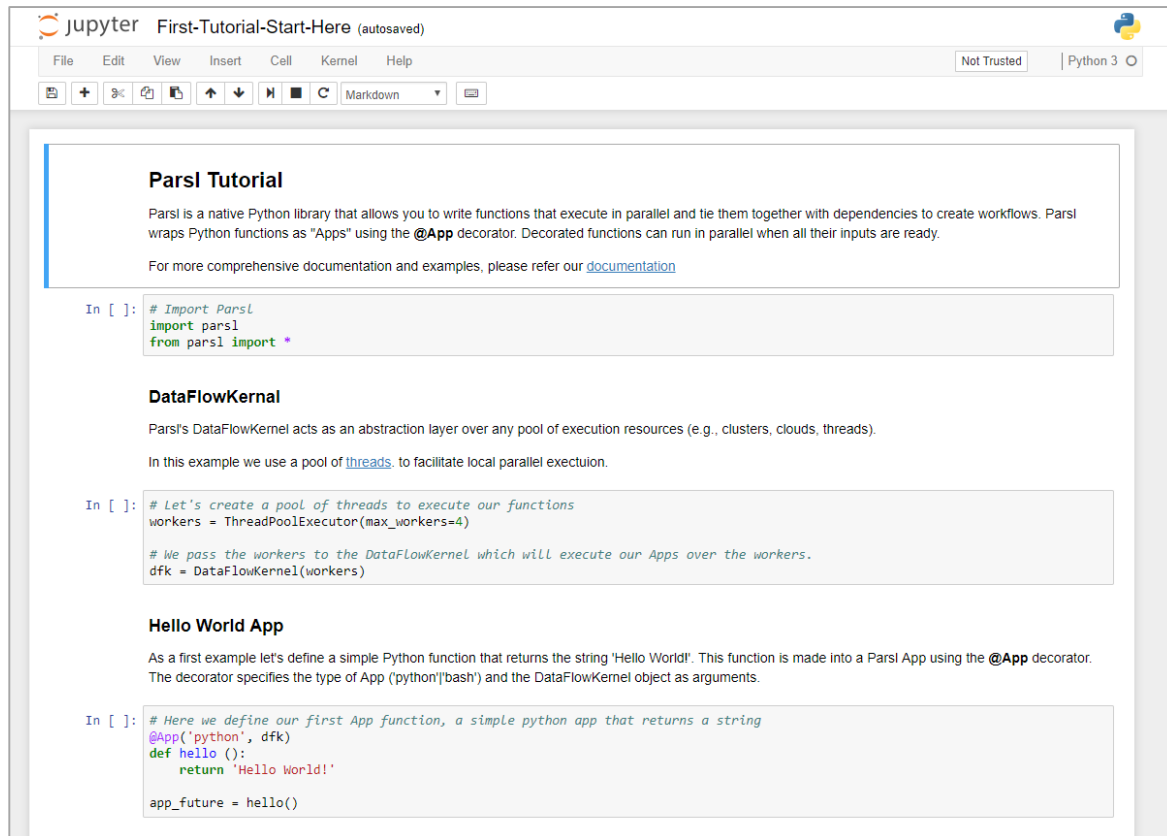




# Interactive supercomputing with Jupyter notebooks

Scalable interactive computing. Run cells, in parallel on large-scale resources

Transparent pass through of authentication tokens in JupyterHub



```
In [ ]: # Import Parsl
import parsl
from parsl import *
```

**DataFlowKernel**

Parsl's DataFlowKernel acts as an abstraction layer over any pool of execution resources (e.g., clusters, clouds, threads).

In this example we use a pool of [threads](#), to facilitate local parallel execution.

```
In [ ]: # Let's create a pool of threads to execute our functions
workers = ThreadPoolExecutor(max_workers=4)

# We pass the workers to the DataFlowKernel which will execute our Apps over the workers.
dfk = DataFlowKernel(workers)
```

**Hello World App**

As a first example let's define a simple Python function that returns the string 'Hello World!'. This function is made into a Parsl App using the **@App** decorator. The decorator specifies the type of App ('python'|'bash') and the DataFlowKernel object as arguments.

```
In [ ]: # Here we define our first App function, a simple python app that returns a string
@app('python', dfk)
def hello ():
    return 'Hello World!'

app_future = hello()
```

# Parsl provides transparent (wide area) data management

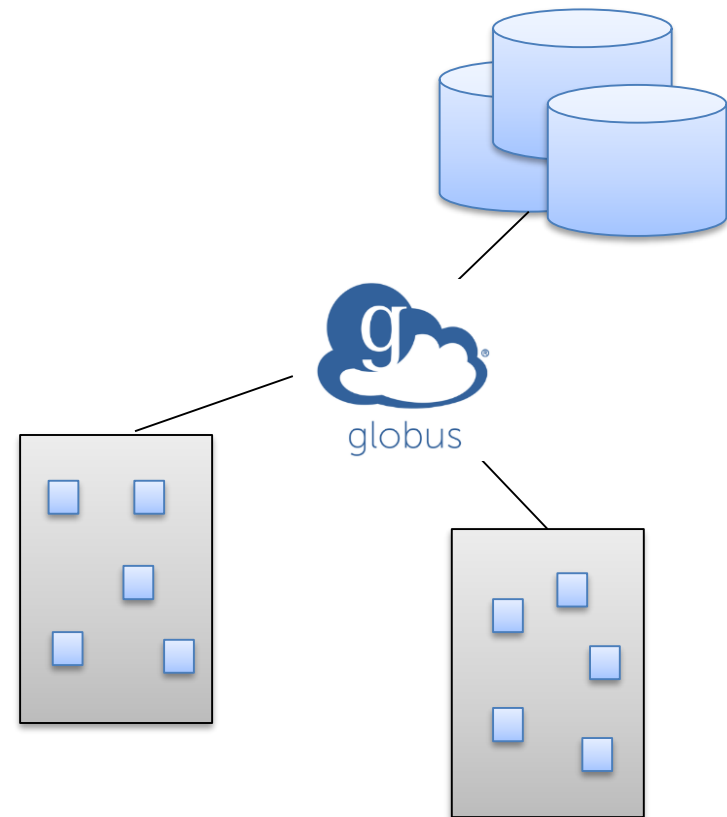
Implicit data movement to/from repositories, laptops, supercomputers, ...

Globus for third-party, high performance and reliable data transfer

- Support for site-specific DTNs

HTTP/FTP direct data download/upload

```
parsl_file =  
    File(globus://EP/path/file)
```



# Parsl tutorial

Running the tutorial online:

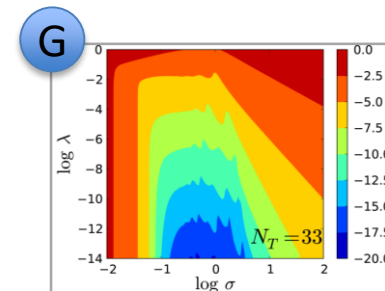
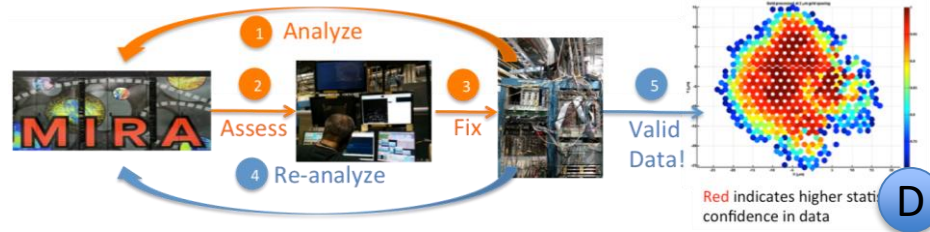
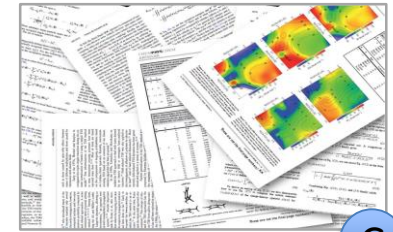
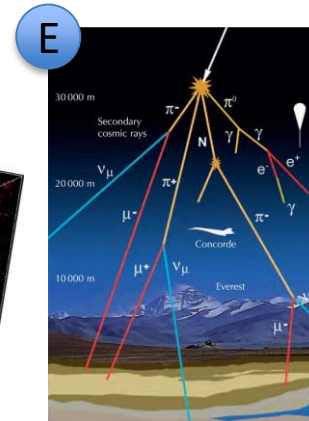
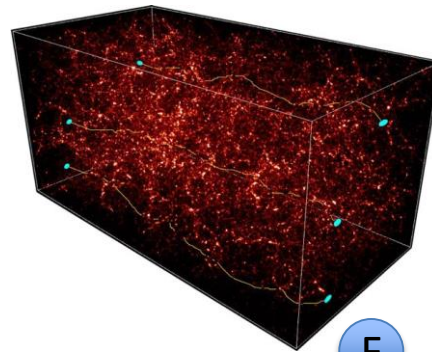
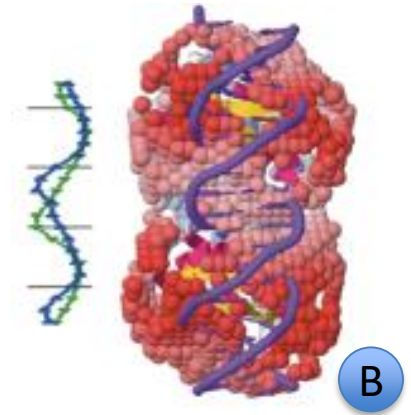
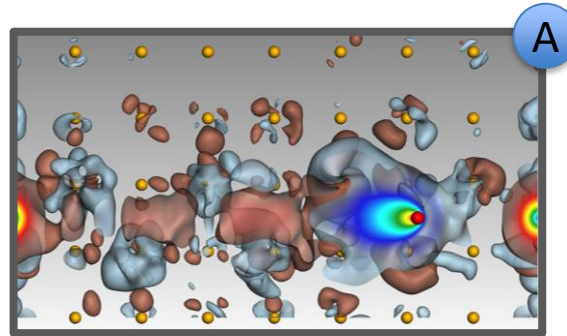
- Binder: <https://mybinder.org/v2/gh/Parsl/parsl-tutorial/master>

Running the tutorial on Blue Waters

- Set up Parsl and download tutorial
  - module load bwpy
  - pip install --user parsl==v0.6.2-a1
  - git clone <https://github.com/Parsl/parsl-tutorial>
  - git checkout bluewaters
- Execution options
  - Download code and run in terminal
  - **Execute notebook on Blue Waters**
    - <https://bluewaters.ncsa.illinois.edu/pythonnotebooks>
  - Execute notebook remotely (e.g., laptop) using Blue Waters

# Large-scale applications using Parsl

- A** Machine learning to predict stopping power in materials
- B** Protein and biomolecule structure and interaction
- C** Information extraction to discovery facts in publications
- D** Materials science at the Advanced Photon Source
- E** Cosmic ray showers as part of QuarkNet
- F** Weak lensing using sky surveys
- G** Machine learning and data analytics



# Summary

Parsl takes a highly successful parallel scripting model and brings it to Python

- No porting of existing scripts to other languages
- Support for both Python and external apps
- Implicit and dynamic dataflow from data dependencies

Applied to numerous MTC and HPC application domains and used on many clusters and supercomputers

Deep integration with growing SciPy ecosystem

*Workflow through implicitly parallel dataflow is productive for applications and systems at many scales, including on highest-end system*

# Questions?

<http://parsl-project.org>

[parsl-project.slack.com](http://parsl-project.slack.com)



THE UNIVERSITY OF  
CHICAGO



U.S. DEPARTMENT OF  
**ENERGY**



**ILLINOIS**  
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN