parsl

Parsl: A Parallel Programming Library for Python

Kyle Chard (chard@uchicago.edu)

On behalf of the Parsl community (including Yadu Babuji, Ben Clifford, Ian Foster, Dan Katz, Zhuozhao Li, Mike Wilde, Anna Woodard)

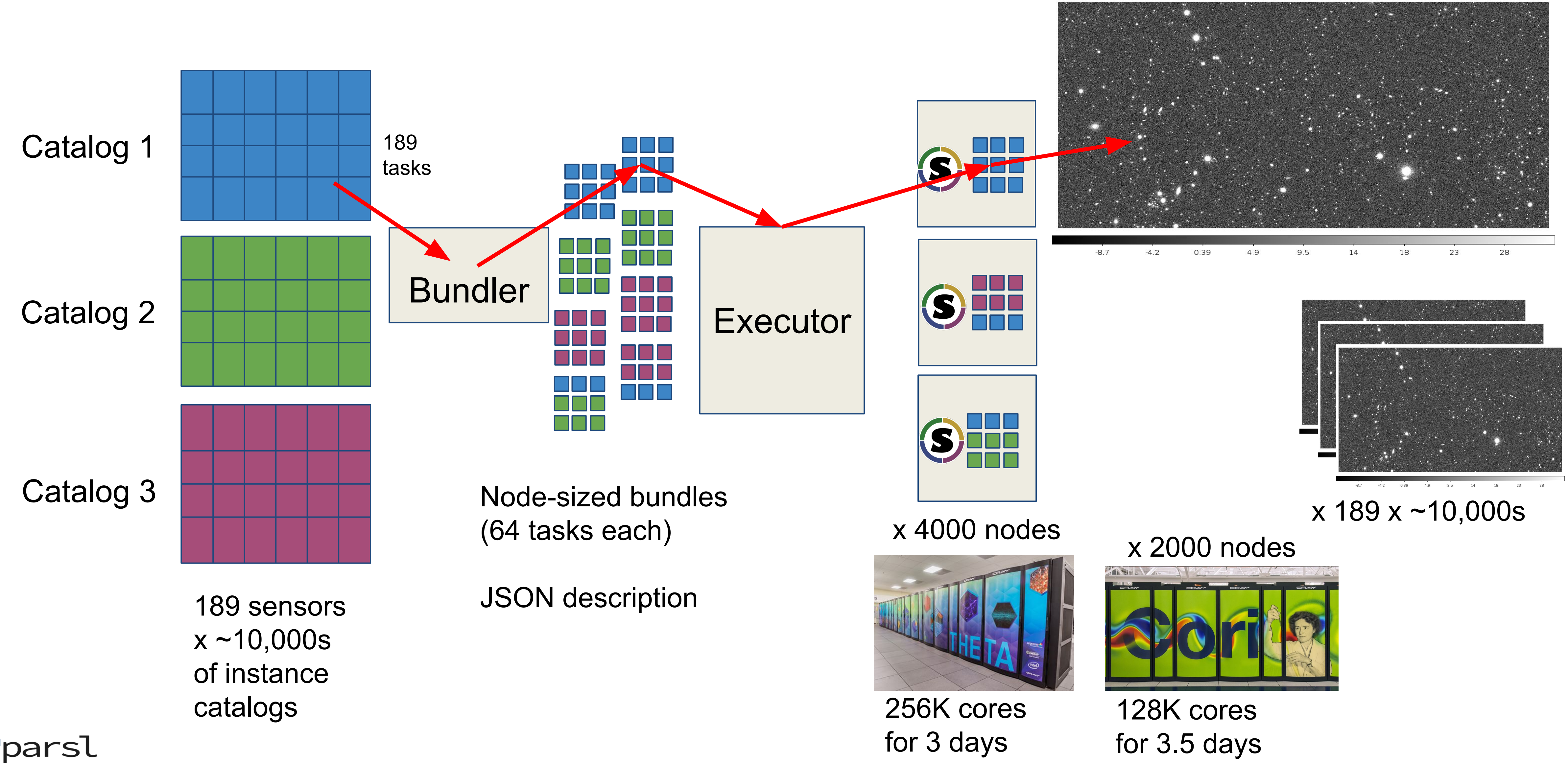
http://parsl-project.org

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

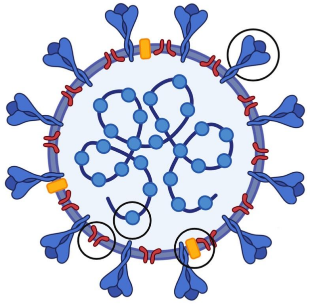
ILLINOIS
NCSA | National Center for Supercomputing Applications

Example workload: simulating images from the Vera C. Rubin Observatory



parsl

Example workload: Applying extreme-scale AI to screen billions of druglike molecules against COVID-19 proteins



CHEMICAL LIBRARY DATABASE

4B known molecules

Enamine

DRUGBANK

BindingDB GDB

eMolecules

cureFFI MOSES

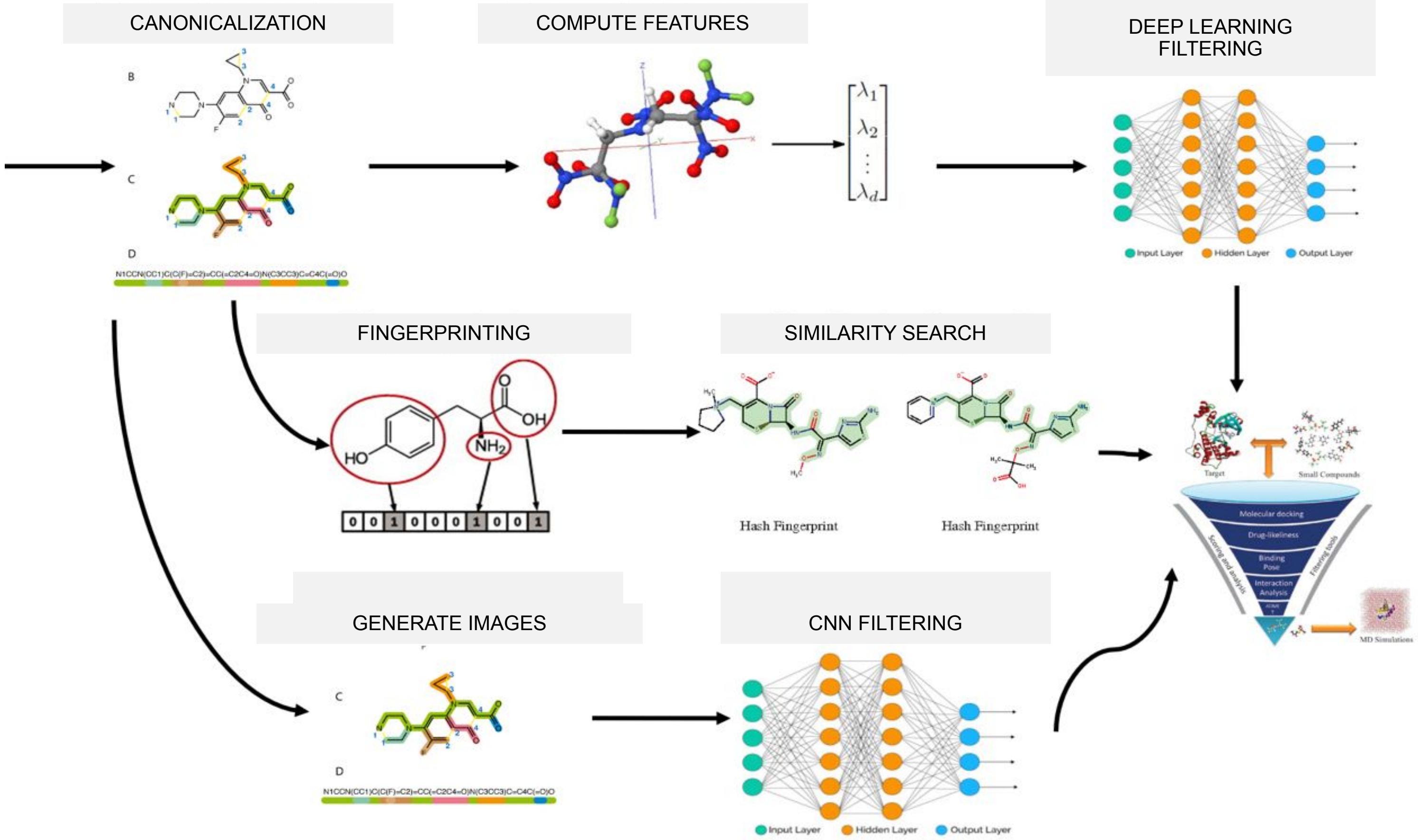
ZINC15

LINCS

AND MORE

SureChEMBL

PubChem



COMPUTING RESOURCES

parsl

Distribution, parallelism, and composition

Parallel and distributed computing is ubiquitous

- Increasing data sizes combined with plateauing sequential processing power

Software is increasingly *assembled* rather than written

- High-level language to integrate and wrap components from many sources

Python (and the SciPy ecosystem) has established itself as one of the most productive and popular environments for research

- Thriving ecosystem of libraries, tools, Jupyter, etc.

Parsl: parallel programming in Python

Apps define opportunities for parallelism

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

```
pip install parsl
```

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

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

```
Hello World!
```

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

```python
@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!
```

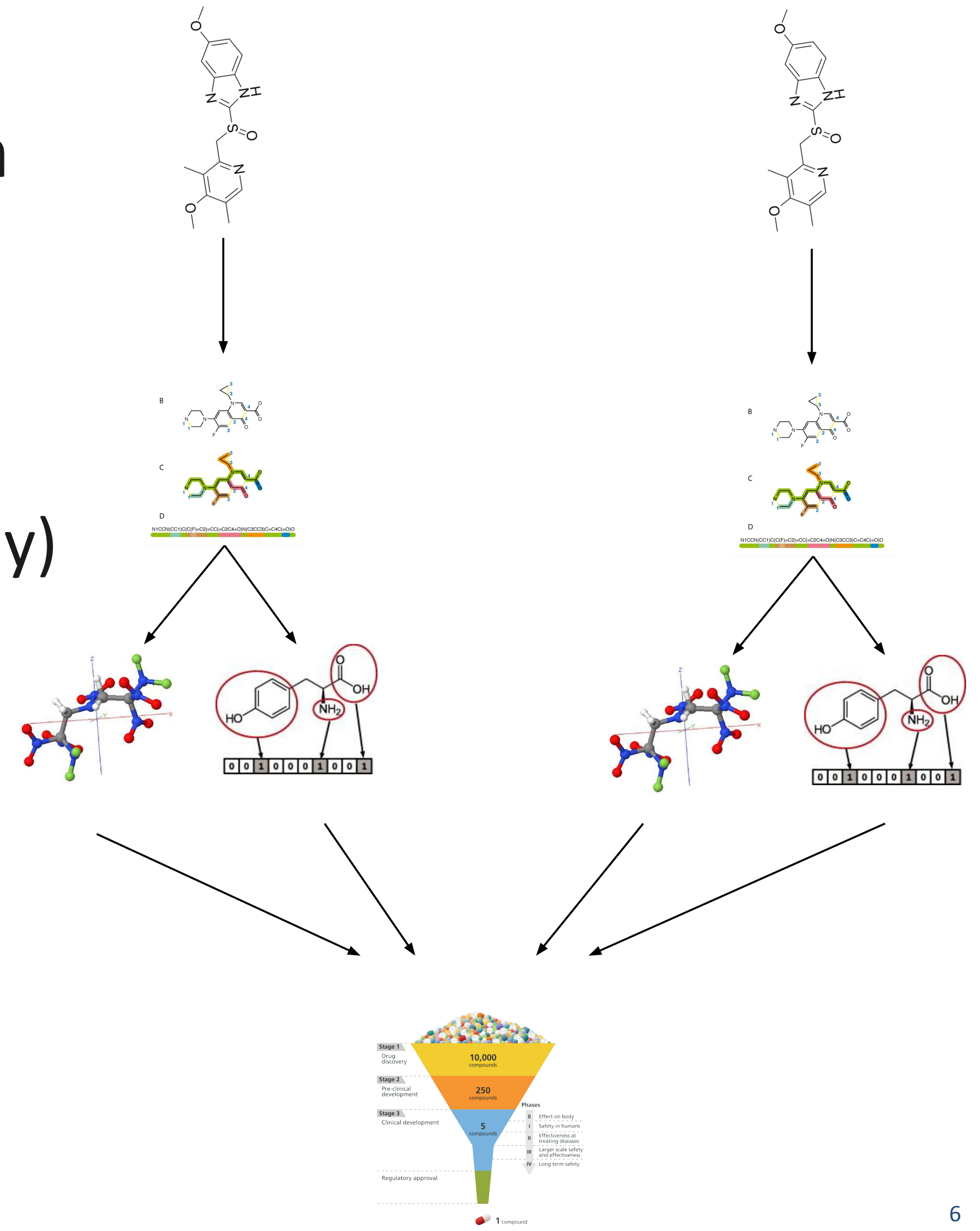
Apps run concurrently respecting dataflow dependencies. Natural parallel programming!

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

Try Parsl: https://parsl-project.org/binder

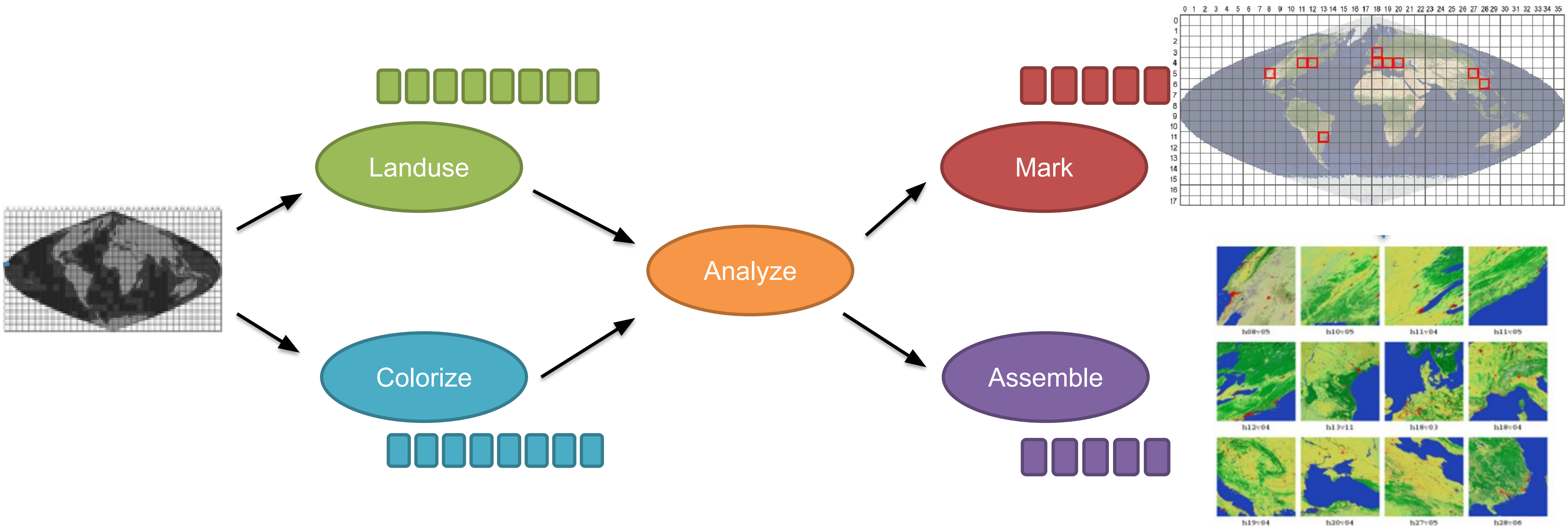
Parsl's dataflow programming model delivers intuitive parallel programming

- Programming paradigm in which program is assembled as a directed graph of data flowing between tasks
- Intuitive way to think about parallelism (tasks run independently when data slice ready)
- Parsl's dataflow model allows data to be passed between Apps
  - Python types and objects
  - Files (local or via HTTP, FTP, or Globus)



parsl

Data-driven example: parallel geospatial analysis



Land-use Image processing pipeline for the MODIS remote sensor

Expressing parallelism using Parsl

1) Wrap the science applications as Parsl Apps:

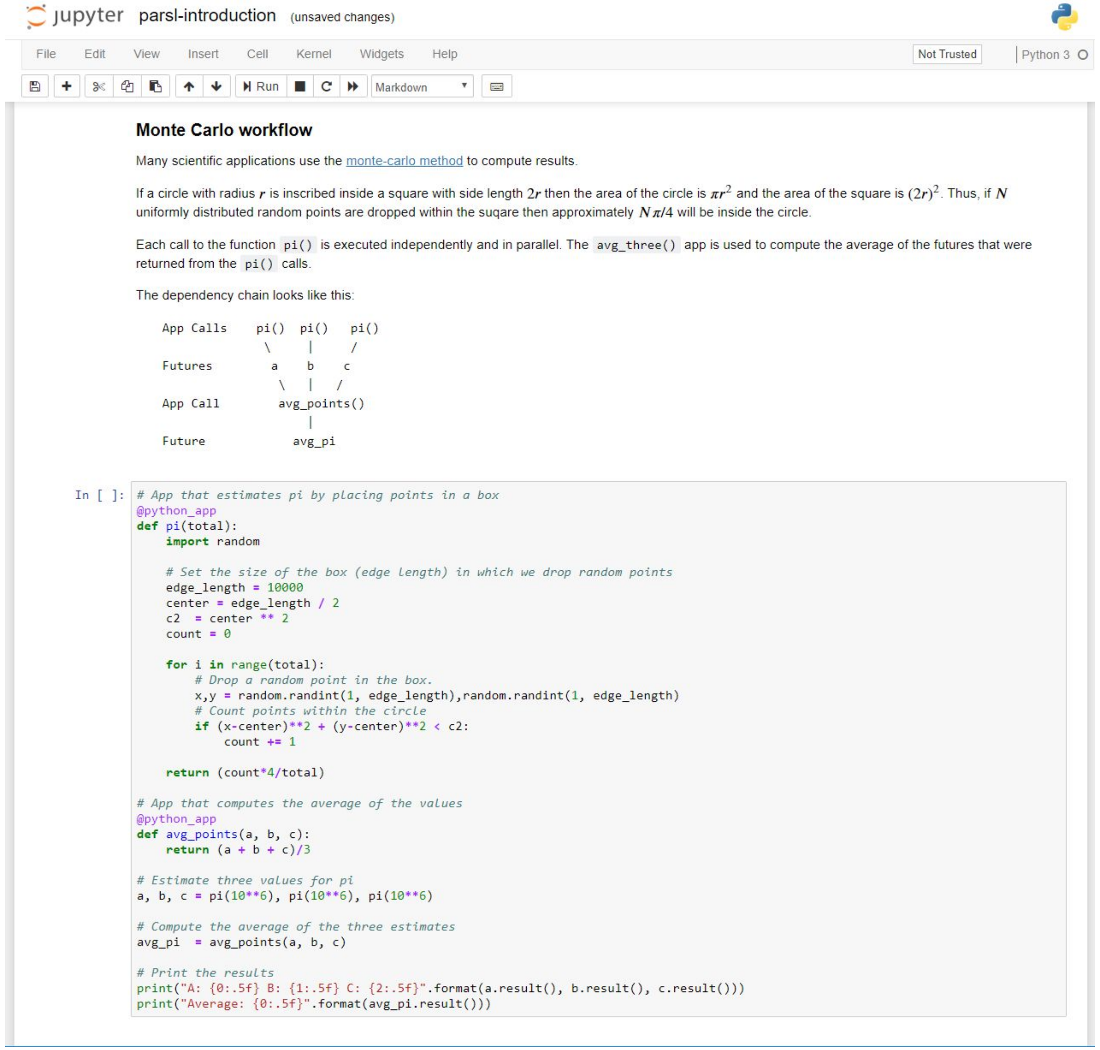
```
@bash_app
def landuse(img, outputs=[]):
  return './landuse_sim.sh {} {}'.format(img, outputs[0])

@python_app
def colorize(img, num_chunks):
  return color_package(img, num_chunks)

@python_app
def analyze(land_chunks, color_chunks):
  return combine(land_chunks, color_chunks)
```

Expressing a many task workflow in Parsl

2) Execute the parallel workflow by calling Apps:

```
lchunks =  []

for i in range (nchunks):
  lchunks.append(landuse(img, outputs=[File('l%s.txt' % i)]))

colored = colorize(img, num_chunks=5)

results = analyze(lchunks, colored)
```

Decomposing parallelism into a dynamic task-dependency graph for distributed execution


jupyter parsl-introduction (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Monte Carlo workflow

Many scientific applications use the monte-carlo method to compute results.

If a circle with radius r is inscribed inside a square with side length $2r$ then the area of the circle is πr^2 and the area of the square is $(2r)^2$. Thus, if N uniformly distributed random points are dropped within the suqare then approximately $N\pi/4$ will be inside the circle.

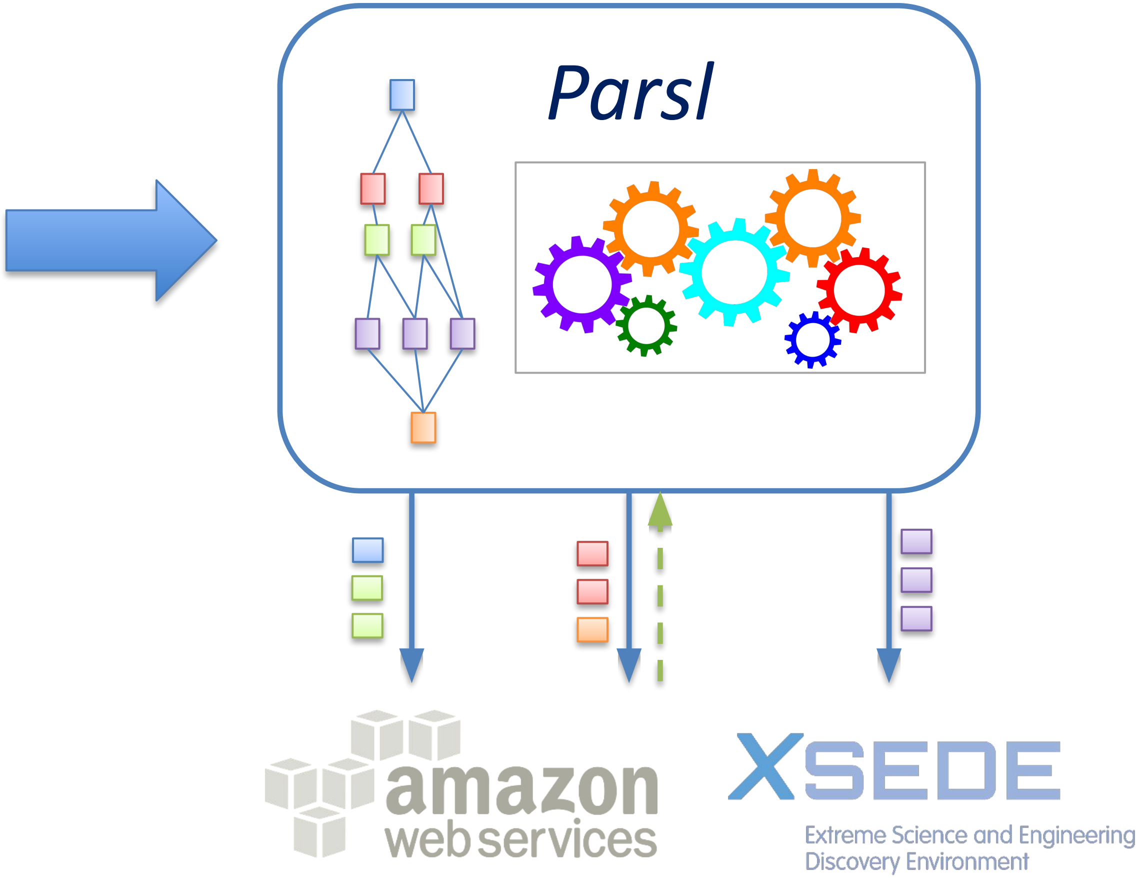
Each call to the function `pi()` is executed independently and in parallel. The `avg_three()` app is used to compute the average of the futures that were returned from the `pi()` calls.

The dependency chain looks like this:

```
App Calls    pi()  pi()   pi()
               \    |     /
Futures         a   b    c
                 \  |   /
App Call        avg_points()
                    |
Future            avg_pi
```

In [ ]:

```
# App that estimates pi by placing points in a box
@python_app
def pi(total):
    import random

    # Set the size of the box (edge length) in which we drop random points
    edge_length = 10000
    center = edge_length / 2
    c2  = center ** 2
    count = 0

    for i in range(total):
        # Drop a random point in the box.
        x,y = random.randint(1, edge_length),random.randint(1, edge_length)
        # Count points within the circle
        if (x-center)**2 + (y-center)**2 < c2:
            count += 1

    return (count*4/total)

# App that computes the average of the values
@python_app
def avg_points(a, b, c):
    return (a + b + c)/3

# Estimate three values for pi
a, b, c = pi(10**6), pi(10**6), pi(10**6)

# Compute the average of the three estimates
avg_pi  = avg_points(a, b, c)

# Print the results
print("A: {0:.5f} B: {1:.5f} C: {2:.5f}".format(a.result(), b.result(), c.result()))
print("Average: {0:.5f}".format(avg_pi.result()))
```



parsl

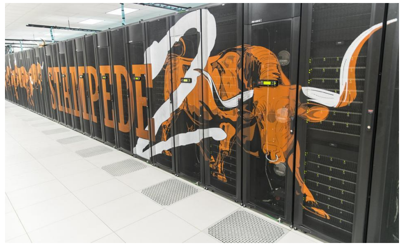
Enabling portable Parsl programs: *providers*

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

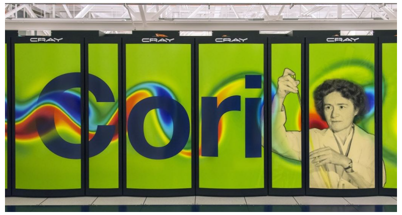
Growing support for various schedulers and cloud vendors

⊟ **Configuration**

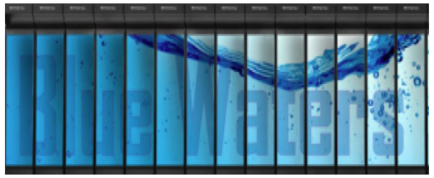
- How-to Configure
- Comet (SDSC)
- Cori (NERSC)
- Stampede2 (TACC)
- Theta (ALCF)
- Cooley (ALCF)
- Swan (Cray)
- CC-IN2P3
- Midway (RCC, UChicago)
- Open Science Grid
- Amazon Web Services
- Ad-Hoc Clusters
- Further help



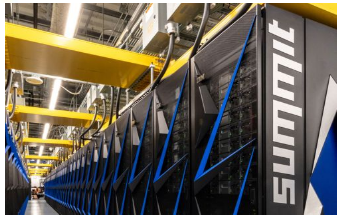
Open Science Grid



amazon web services



Google Cloud



XSEDE

parsl

Separation of code and execution

sample_configs.py

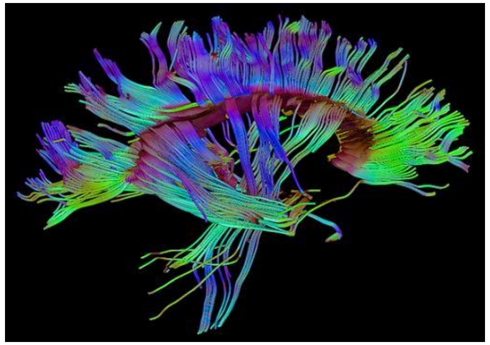
```
# ... imports

threads_config = Config(
    executors=[ThreadPoolExecutor()]
)

cori_config = Config(
    executors=[
        HighThroughputExecutor(
            label='Cori_HTEX_multinode',
            provider=SlurmProvider(
                'debug',   # Partition / QOS
                nodes_per_block=2,
                walltime="00:20:00",
                launcher=SrunLauncher()
            ))
    ])
```

runner.py

```
import parsl
import os
from sample_configs import threads_config, cori_config

if os.environ.get('PIPELINE_ENV', 'test'):
    parsl.load(threads_config)
else:
    parsl.load(cori_config)

#... rest of the pipeline...
```

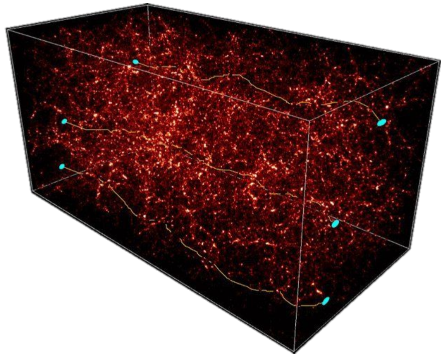
Choose execution environment at runtime. Parsl will direct tasks to the configured execution environment(s).

Parsl implements a Python's Concurrent.futures *executor* (runtime) interface



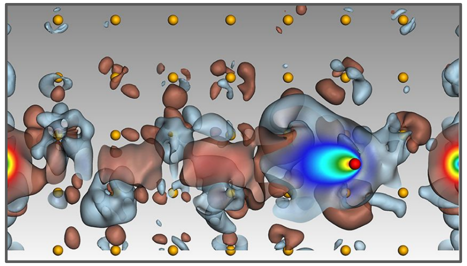
High-throughput executor (HTEX)

- Pilot job-based model with multi-threaded manager deployed on workers
- Designed for ease of use, fault-tolerance, etc.
- <2000 nodes (~60K workers), Ms tasks, task duration/nodes > 0.01



Extreme-scale executor (EXEX)*

- Distributed MPI job manages execution. Manager rank communicates workload to other worker ranks directly
- Designed for extreme scale execution on supercomputers
- >1000 nodes (>30K workers), Ms tasks, >1m task duration



Low-latency Executor (LLEX)*

- Direct socket communication to workers, fixed resource pool, limited features
- 10s nodes, <1M tasks, <1m tasks

Others: WorkQueue, RADICAL-Cybertools, Flux

Parsl executors scale to 256K concurrent workers

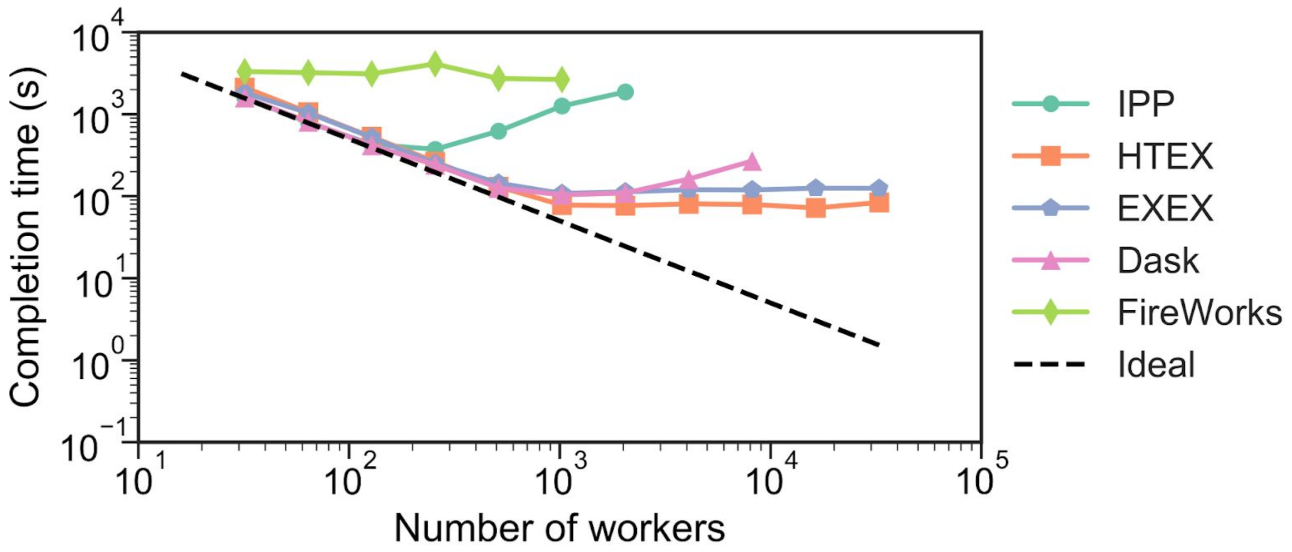
HTEX and EXEX outperform other Python-based approaches

Parsl scales to more than 250K workers (8K nodes) and ~2M tasks

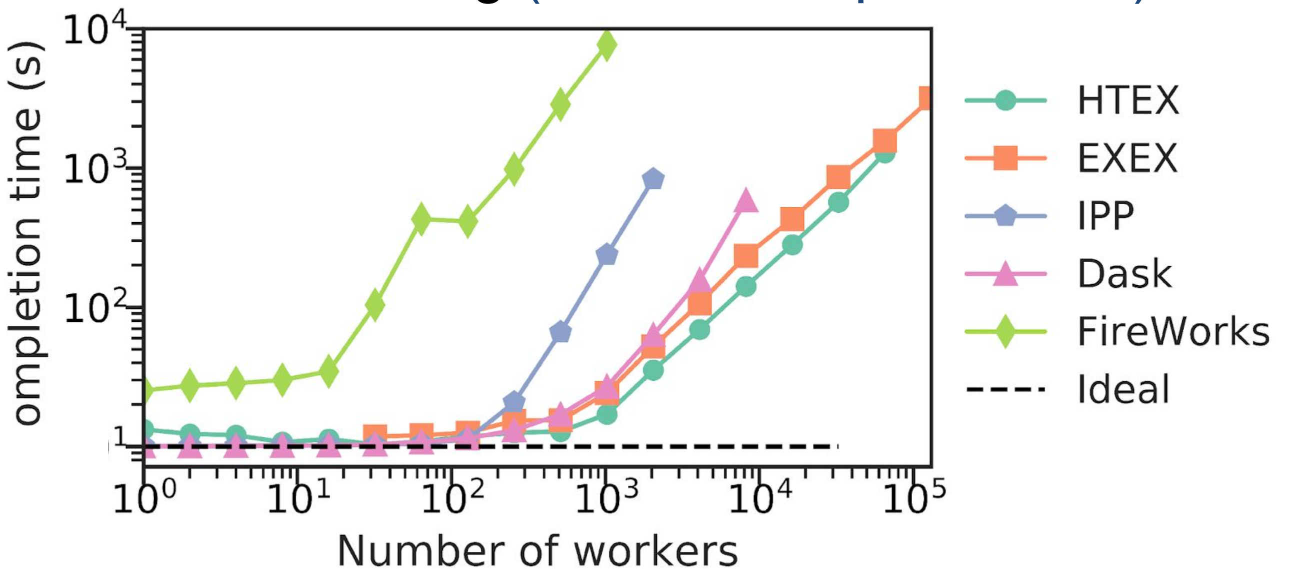
Framework	Maximum # of workers†	Maximum # of nodes†	Maximum tasks/second‡
Parsl-IPP	2048	64	330
Parsl-HTEX	65 536	2048*	1181
Parsl-EXEX	262 144	8192*	1176
FireWorks	1024	32	4
Dask distributed	4096	128	2617

Babuji et.al. "Parsl: Pervasive Parallel Programming in Python." ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC). 2019.

Strong scaling (50K 1s tasks)

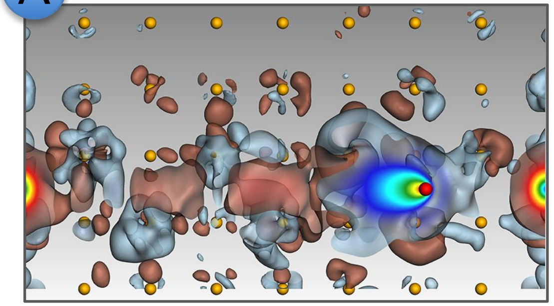


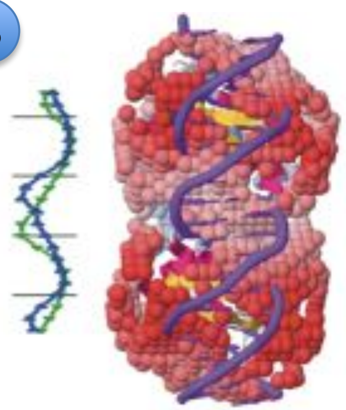
Weak scaling (10 1s tasks per worker)

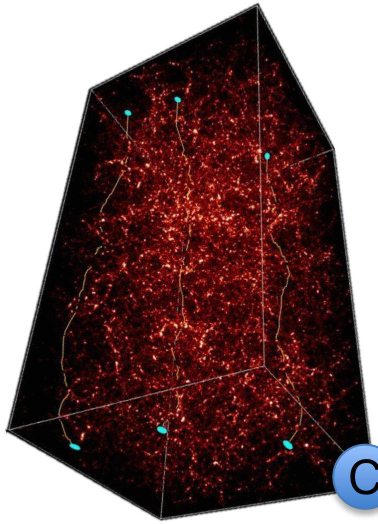


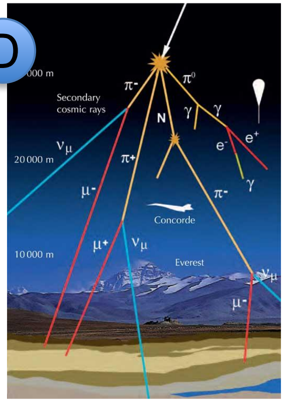
Parsl is being used in a wide range of scientific applications

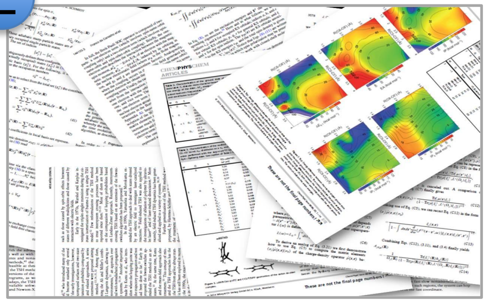
- **A** Machine learning to predict stopping power in materials
- **B** Protein and biomolecule structure and interaction
- **C** LSST simulation and weak lensing using sky surveys
- **D** Cosmic ray showers in QuarkNet
- **E** Information extraction to classify image types in papers
- **F** Materials science at the Advanced Photon Source
- **G** Machine learning and data analytics in materials

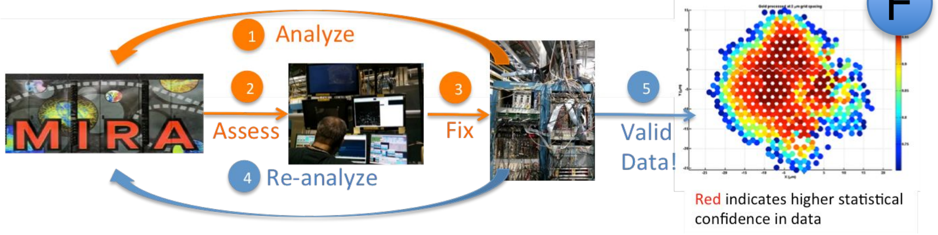


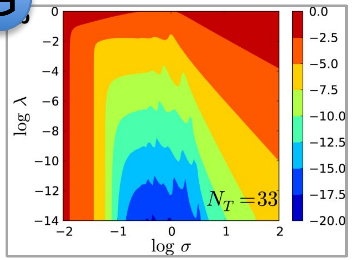








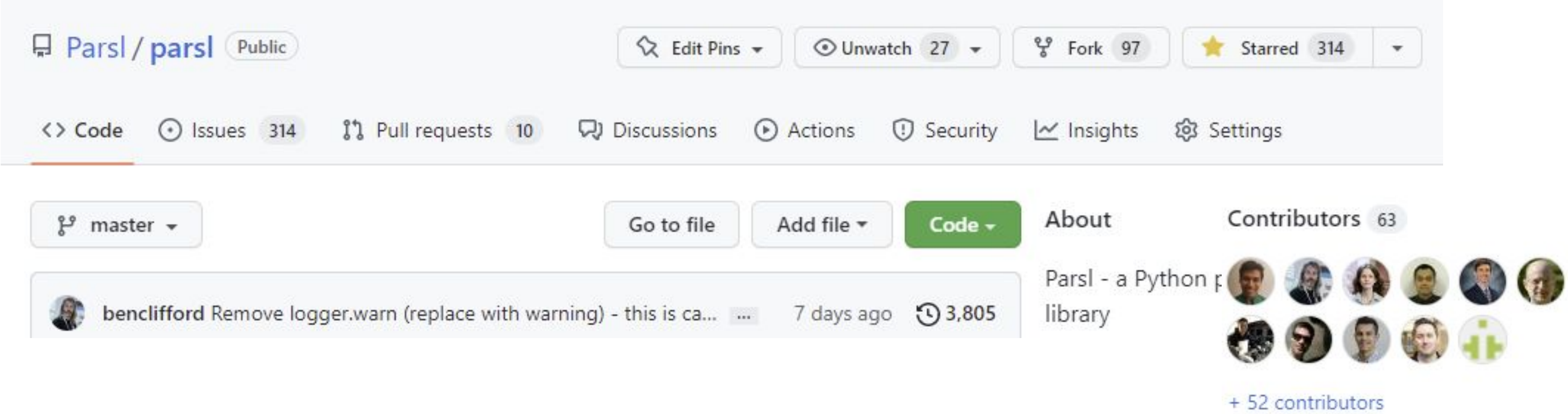




https://parsl-project.org/parslfest.html

parsl

Parsl is an open-source Python community (parsl-project.org)



funcX: Parsl as a service for remote computing

Common Parsl use case: I want to run my computation on one or more remote clusters, clouds, supercomputers from my PC

Cloud-hosted managed compute service built on Parsl

FuncX: Fire-and-forget remote function execution

funcX Service:

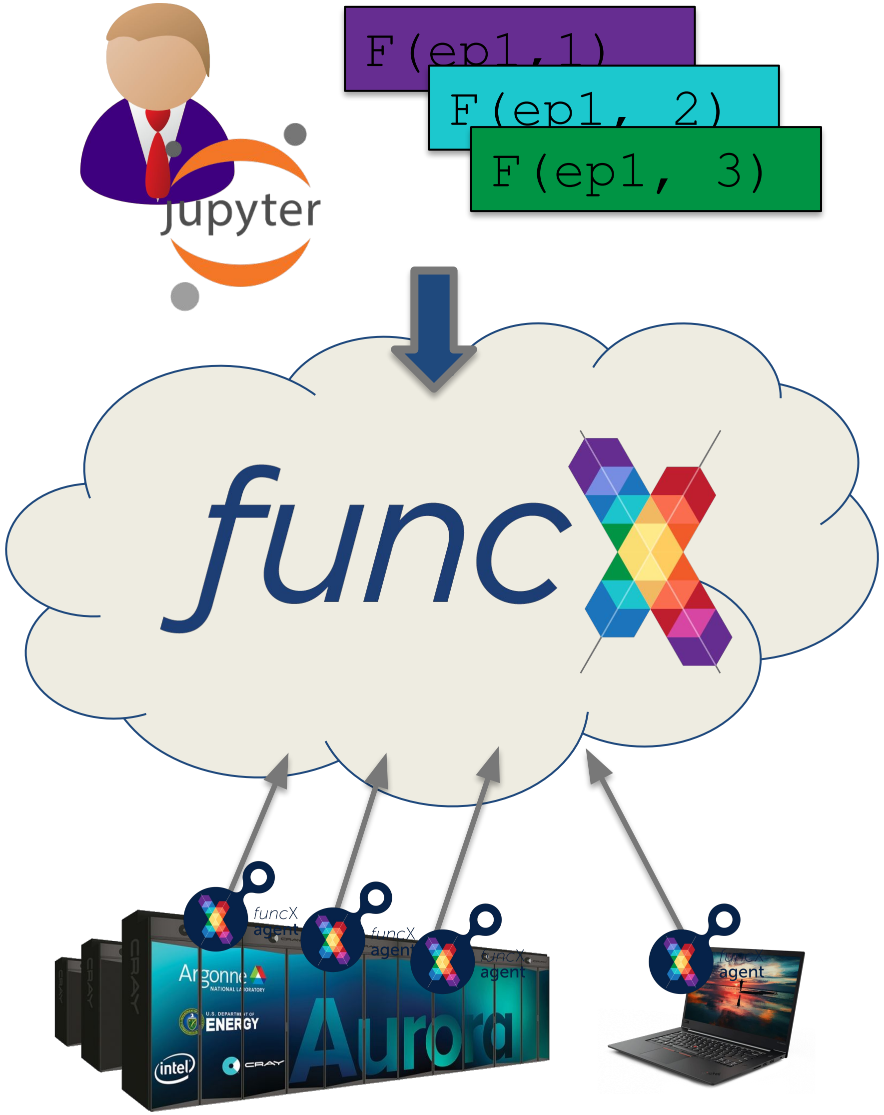
- Single reliable cloud service (REST)
- Register, share, run functions
- Fire-and-forget execution: funcX will manage execution, store results in the cloud, handle errors, etc.

Endpoints:

- User-deployed and managed
- Dynamically provision resources, deploy containers, execute functions, catch exceptions, etc.
- Exploit local architecture/accelerators

Choose where to execute functions

- Closest, cheapest, fastest, accelerators …



funcx.org

Parsl provides productive, safe, scalable, and flexible parallelism in Python

Productive: Python with minimal new constructs (integrated with the growing SciPy ecosystem and other scientific services)

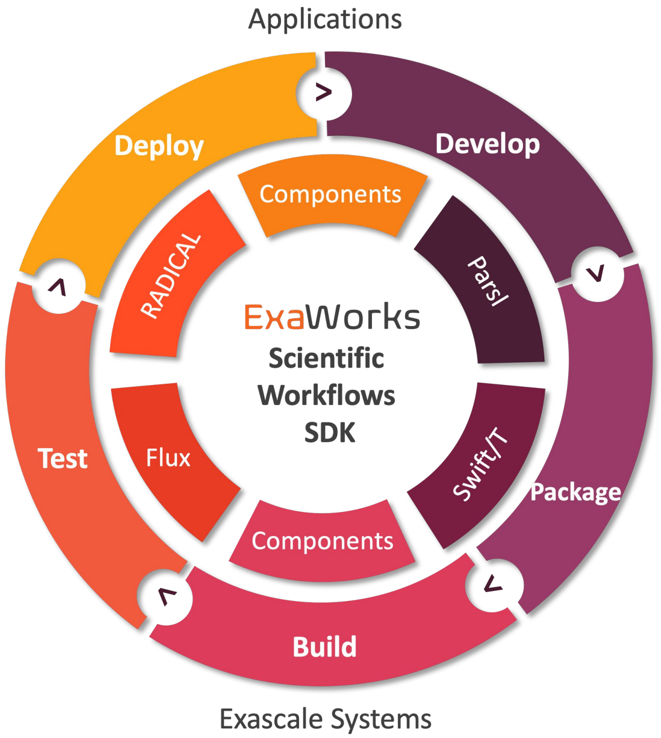
Safe: deterministic parallel programs through immutable input/output objects, dependency task graph, etc.

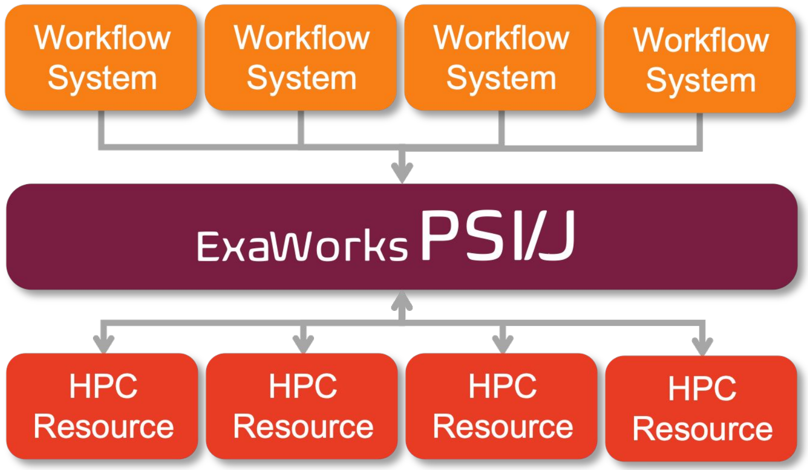
Scalable and portable: efficient execution from laptops to the largest supercomputers

Flexible: programs composed from existing components and then applied to different resources/workloads

ExaWorks: ECP Workflows Project

- Community-based project to support workflows in ECP
  - Users, workflow developers, facilities, vendors
- Technical development
  - ExaWorks SDK: packaged and compatible workflows **components**
  - PSI/J: Asynchronous Python library for scheduler portability
- Community development
  - Organizing a series of summits to bring the workflows community together
  - https://exaworks.org/summit.html
- Contributing to the Workflows Community Initiative (https://workflows.community)





Questions?

parsl-project.org

parsl-project.org/binder

amazon web services
NSF
THE UNIVERSITY OF CHICAGO
ILLINOIS
NCSA | National Center for Supercomputing Applications
U.S. DEPARTMENT OF ENERGY
Argonne NATIONAL LABORATORY