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 and many others)

http://parsl-project.org

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

ILLINOIS
NCSA | National Center for Supercomputing Applications

Motivation: The modern research computing landscape

Software is increasingly *assembled* rather than written

- High-level language to integrate components from many sources

Parallel and distributed computing is ubiquitous

- Increasing data sizes & plateauing sequential processing

Resources are increasingly heterogeneous (distributed)

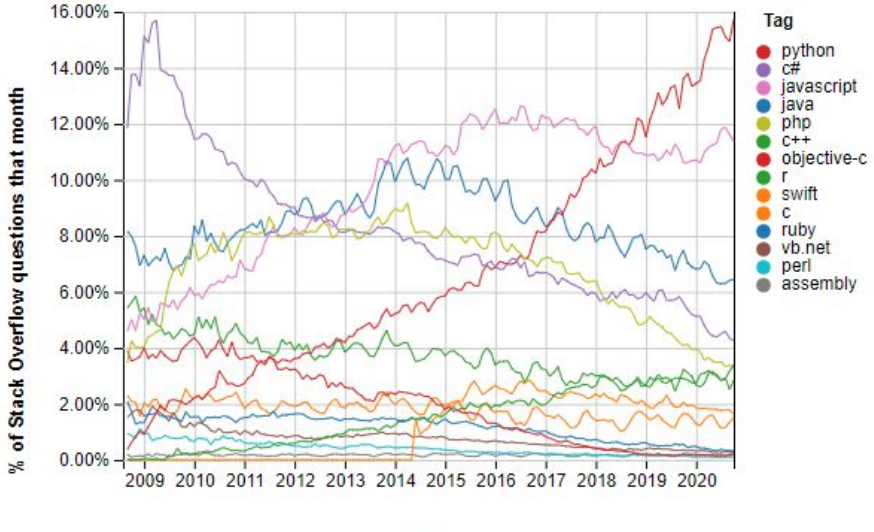
- Application components best run in different places

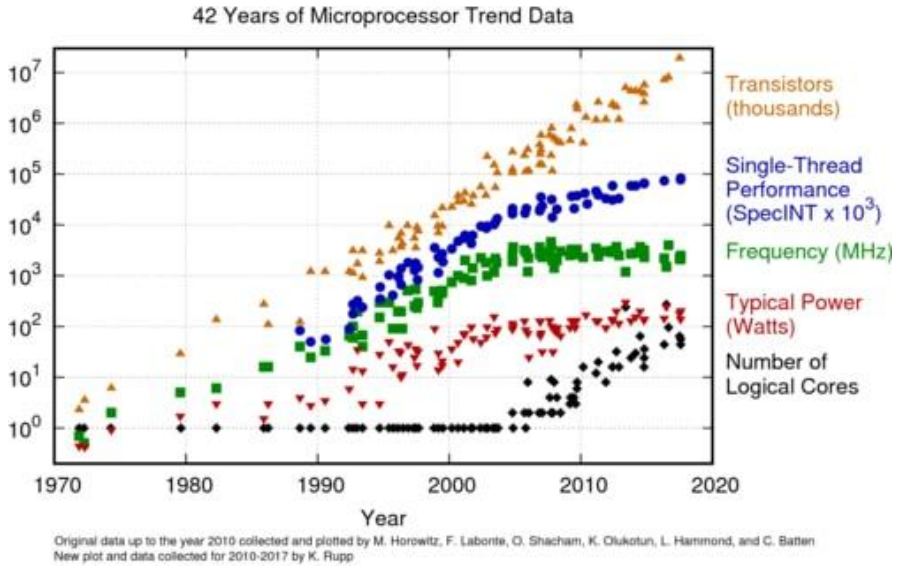
Python (& SciPy ecosystem) de facto standard language

- Libraries, tools, Jupyter, etc.

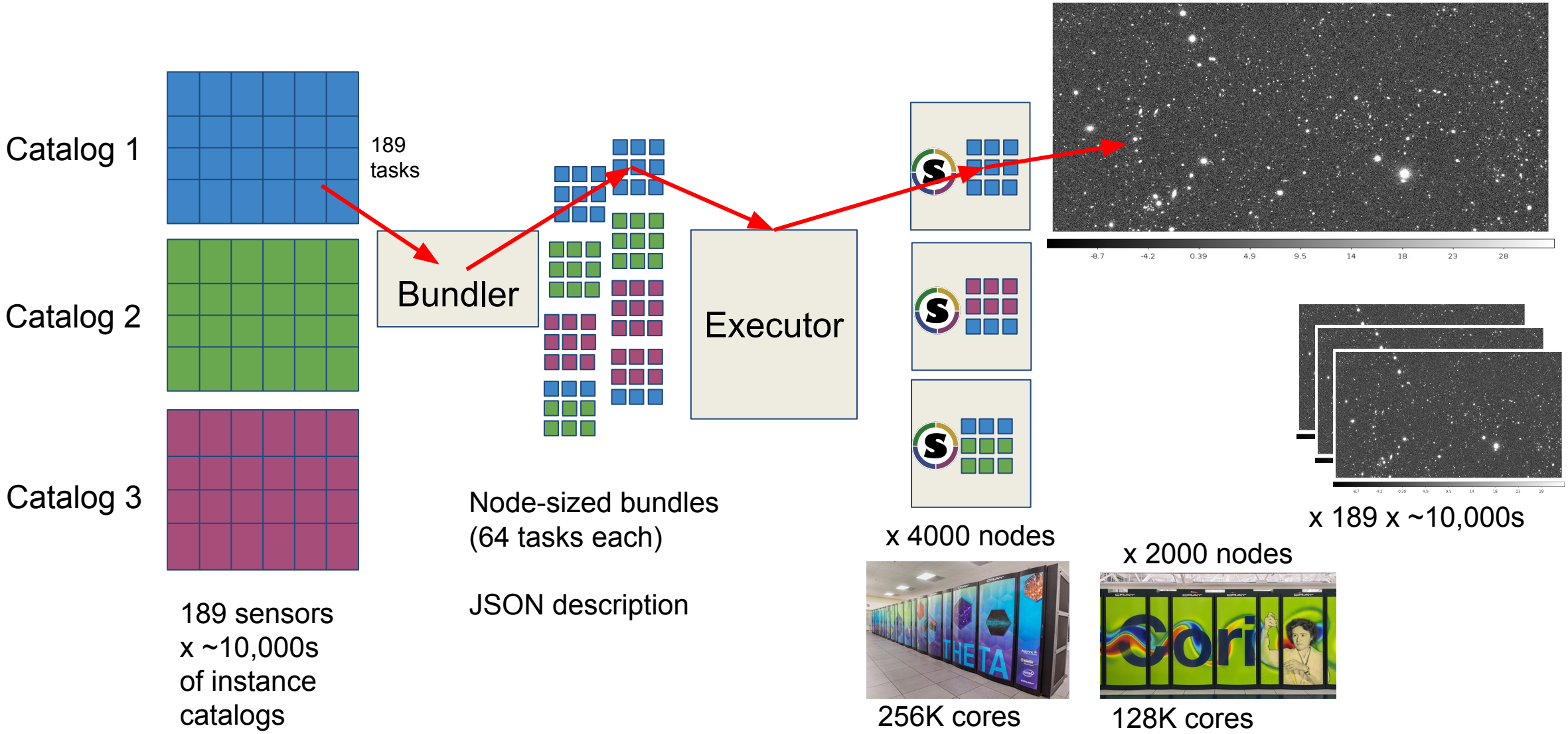
Parsl allows for the natural expression of parallelism in Python

funcX enables fire-and-forget remote and distributed execution of Python functions

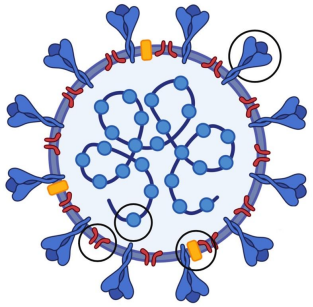




Example workload: simulating images from the Vera Rubin Observatory



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

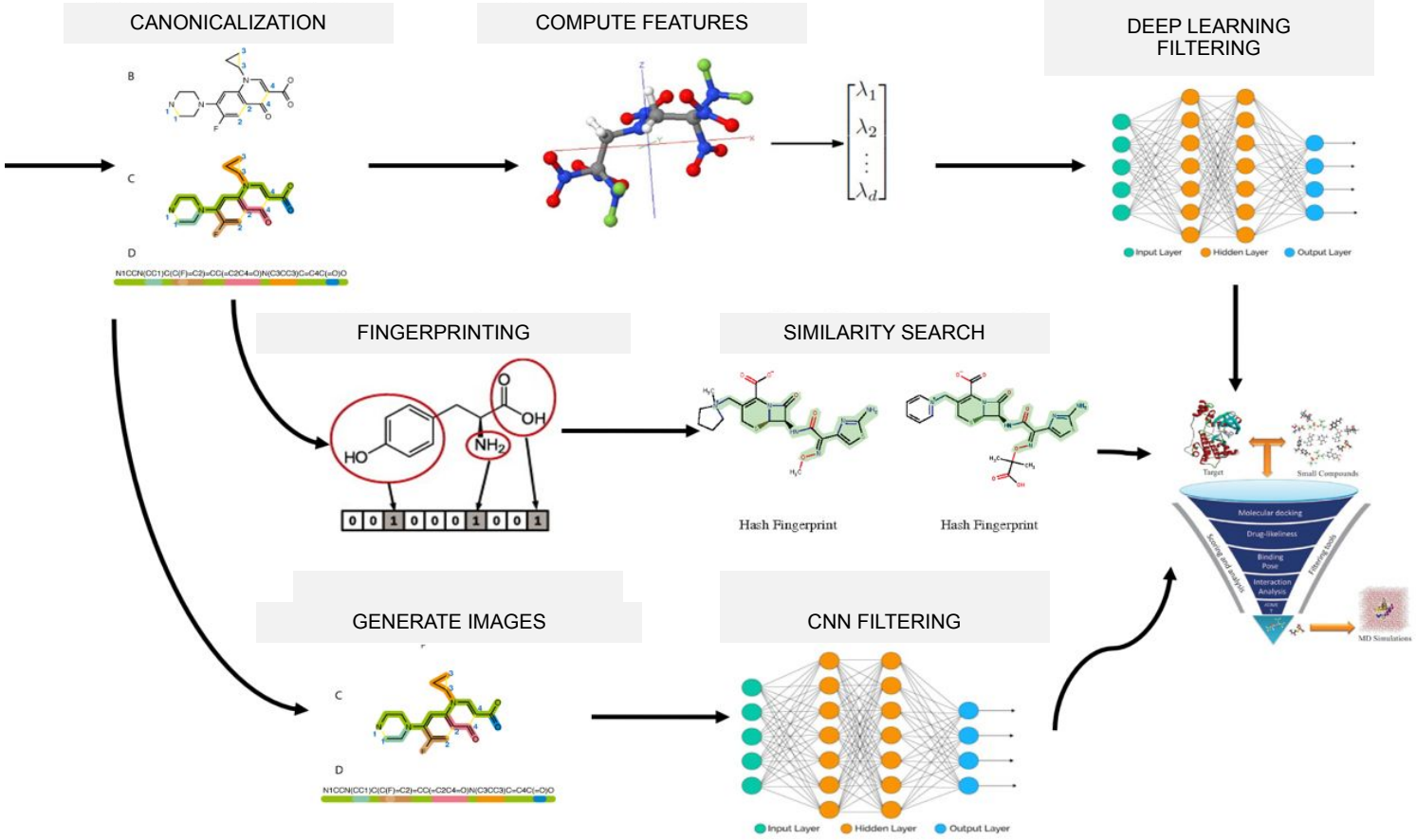
ZINC15

LINCS

AND MORE

SureChEMBL

PubChem



COMPUTING RESOURCES

parsl

Parsl: parallel programming in Python

Apps define opportunities for parallelism

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

```
pip install parsl
```

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

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

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

```
Hello World!
```

Apps run concurrently respecting dataflow dependencies. Natural parallel programming!

```bash
@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())
```

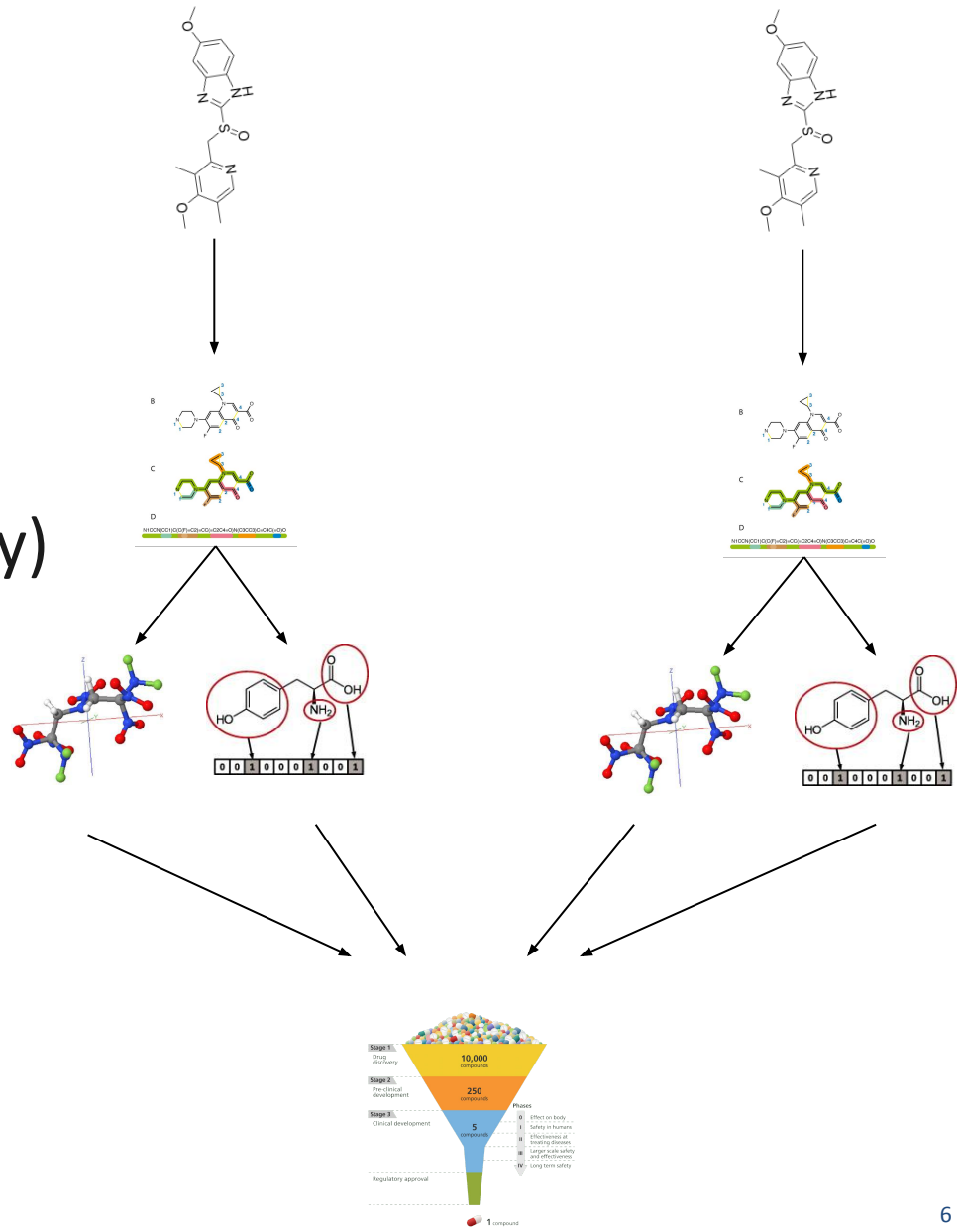
```
Hello World!
```

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

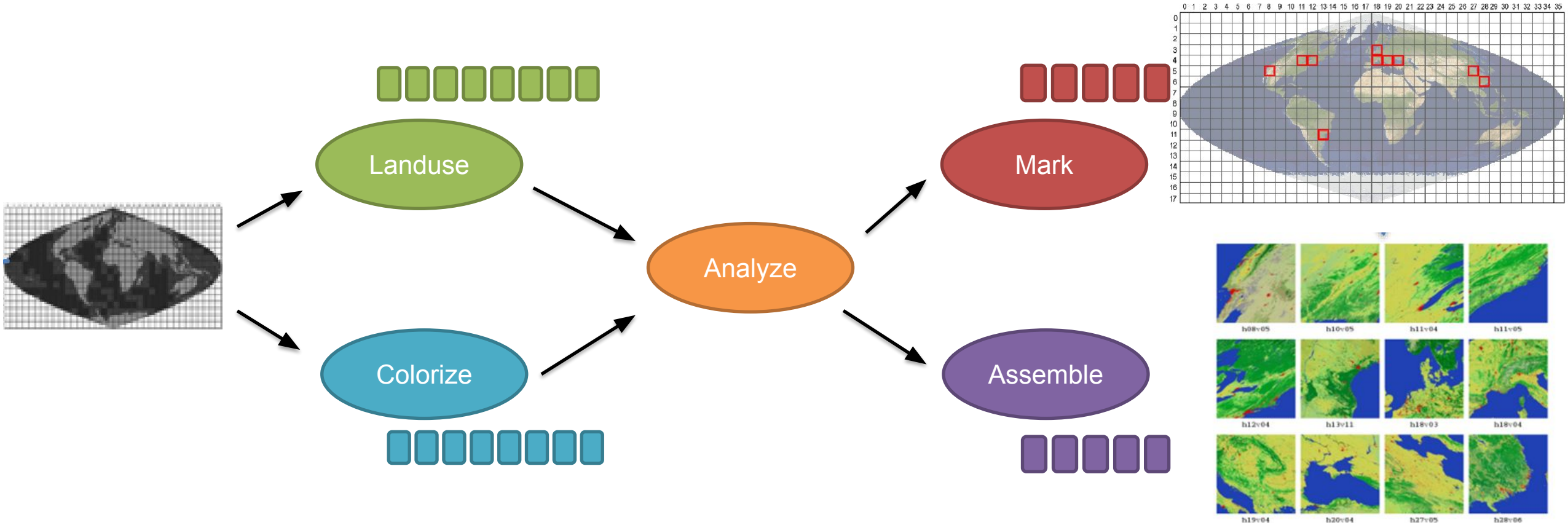
Try Parsl: https://parsl-project.org/binder

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)



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

Run Markdown

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.

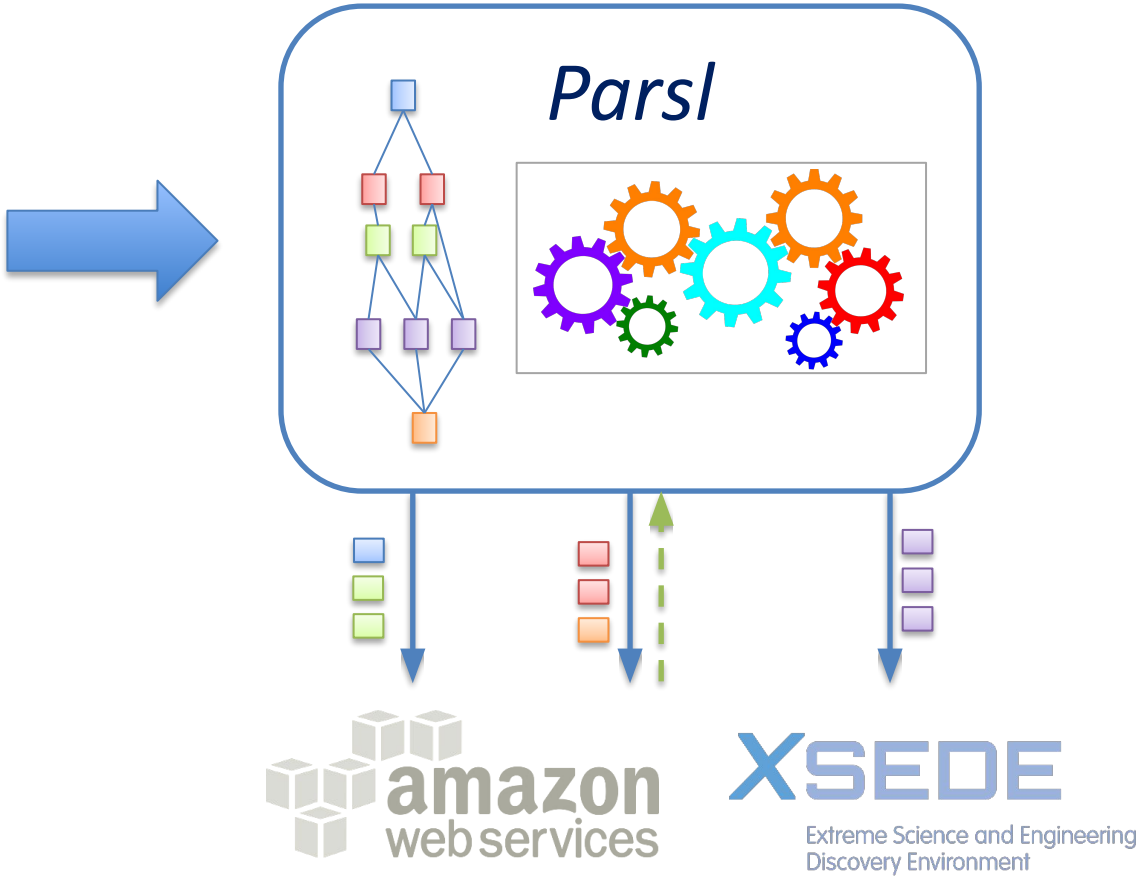
Each call to the function `pi()` is executed independently and in parallel. The `avg_three()` app is used to compute the average of the futures that were returned from the `pi()` calls.

The dependency chain looks like this:

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

In [ ]:

```python
# 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 programs can be executed in different ways on different systems

Executors (`concurrent.futures.Executor` interface)

- HTEX
- Work Queue
- Flux
- EXEX
- RADICAL-Cybertools
- funcX
- IPyParallel

Production

Prototype

Deprecated

Providers

Slurm	LSF	GridEngine	Kubernetes	AWS
PBS	Cobalt	HTCondor	Google	Ad hoc

Separation of code and execution

sample_configs.py

```python
# ... 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

```python
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 executors scale to 256K concurrent workers

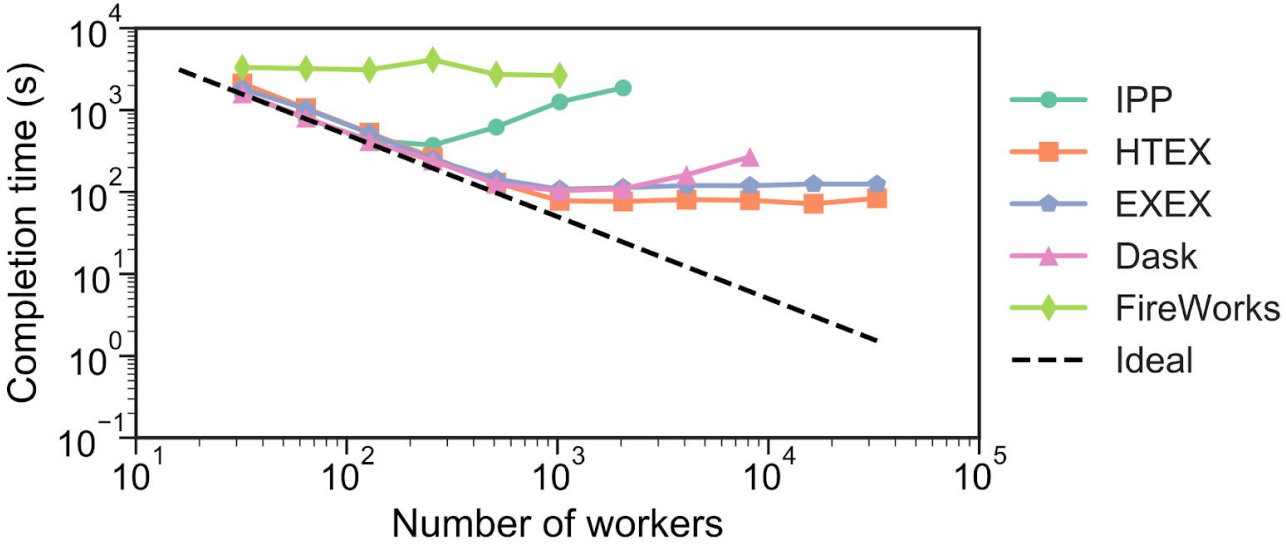
HTEX and EXEX outperform other Python-based approaches

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

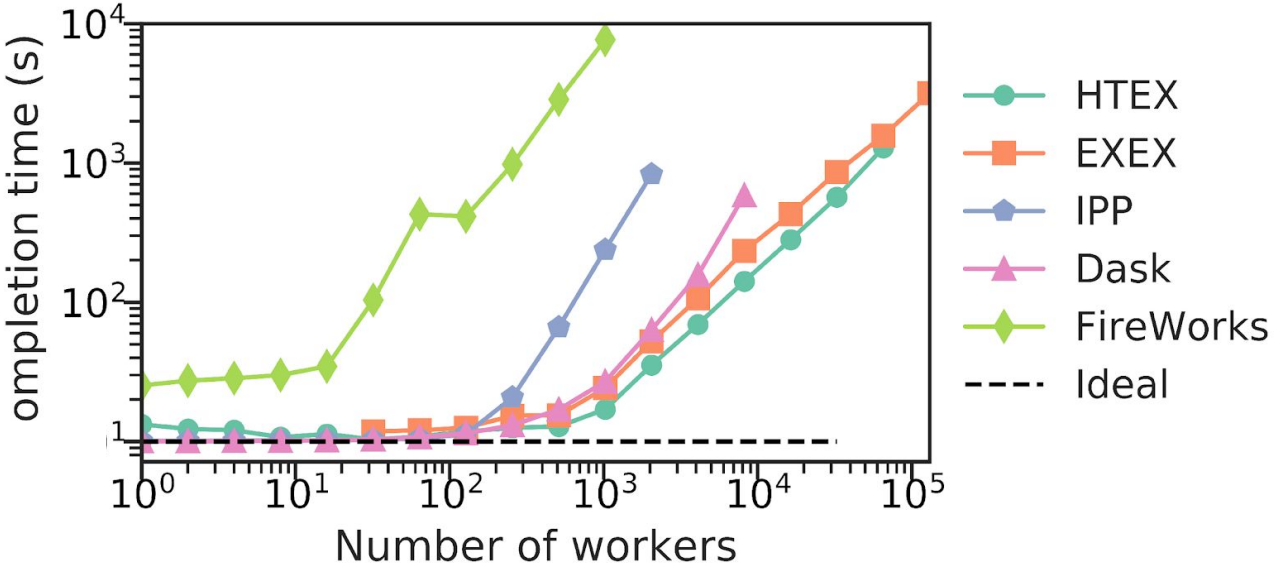
Framework	Maximum # of workers†	Maximum # of nodes†	Maximum tasks/second‡
Parsl-IPP	2048	64	330
Parsl-HTEX	65 536	2048*	1181
Parsl-EXEX	262 144	8192*	1176
FireWorks	1024	32	4
Dask distributed	4096	128	2617

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)



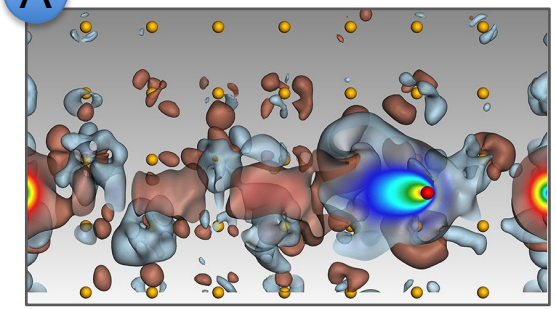
Weak scaling (10 1s tasks per worker)



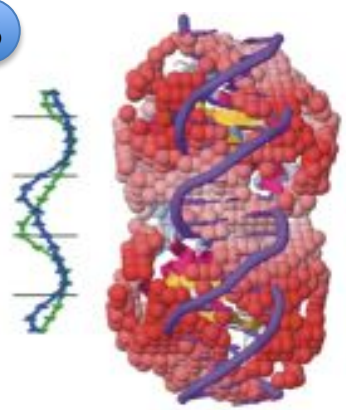
Parsl is being used in a wide range of scientific applications

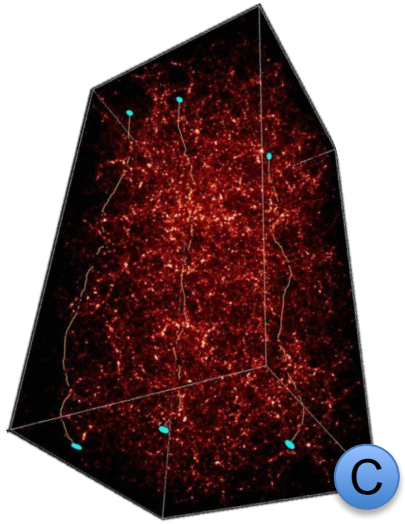
- **A** Machine learning to predict stopping power in materials
- **B** Protein and biomolecule structure and interaction
- **C** LSST simulation and weak lensing using sky surveys
- **D** Cosmic ray showers in QuarkNet
- **E** Information extraction to classify image types in papers
- **F** Materials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

A



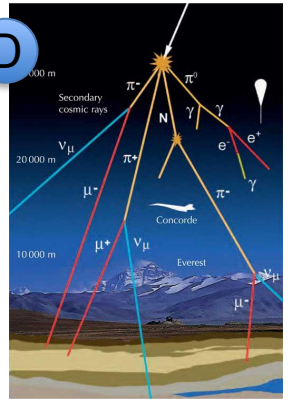
B



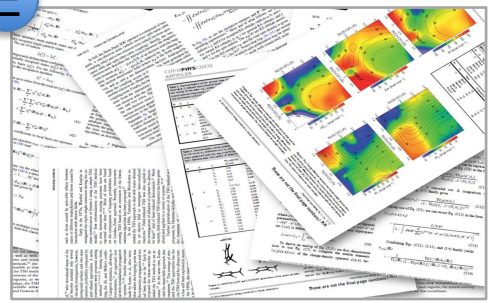


C

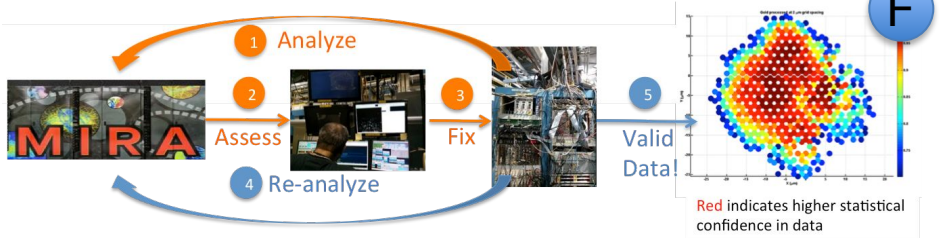
D



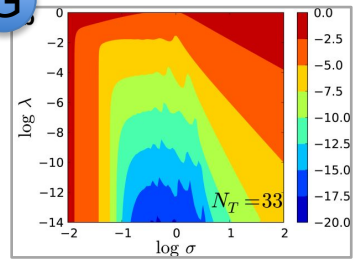
E



F



G



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

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

Parsl / parsl Public

Edit Pins | Unwatch 27 | Fork 108 | Starred 341

Code | Issues 300 | Pull requests 8 | Discussions | Actions | Security | Insights | Settings

master | 111 branches | 42 tags | Go to file | Add file | Code

WardLT Support using spawn instead of fork for HTEx (#2433) ... dfcbff8 6 days ago 3,977 commits

.github Follow up to get release automation point at Pypi (#2448) 8 days ago

Contributors 70

+ 59 contributors

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

funcX

Cloud-hosted managed compute service built on Parsl

funcX: Federated Function-as-a-Service

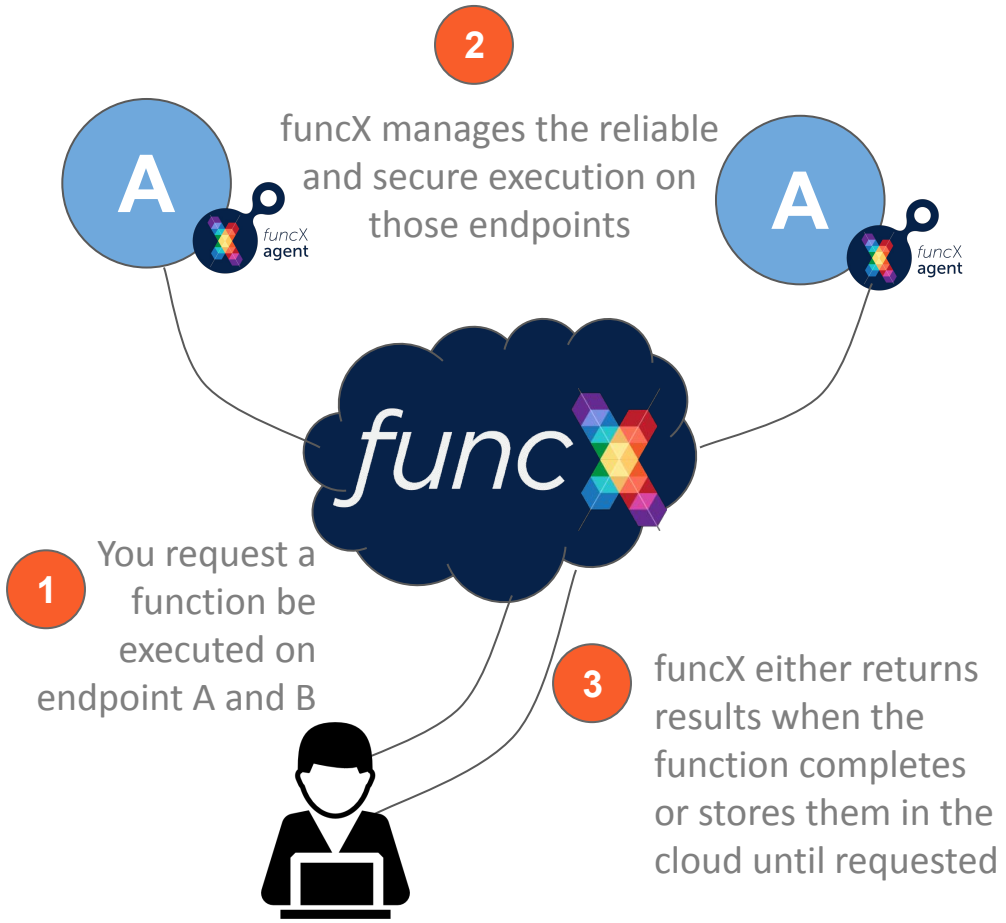
One of the biggest challenges with Parsl is the need to coordinate work over several resources

- Where do you host the Parsl process? how do you reliably connect to remote resources? how do you manage data between them?

funcX servicifies Parsl:

- funcX endpoints (building on Parsl) enable scalable execution of tasks
- funcX service provides reliable management of tasks across endpoints

funcX leverages the Globus security model



Try funcX: https://funcx.org/binder

Transform laptops, clusters, clouds into function serving endpoints

- Python-based agent (pip or Conda) installable in user space
- Elastically provisions resources from local, cluster, kubernetes, or cloud system (using Parsl)
- Manages concurrent execution on provisioned resources
- Optionally manages execution in containers
- Share endpoints with collaborators

```
$ pip install funcx-endpoint

$ funcx-endpoint configure myep

$ funcx-endpoint start myep
```

Execute tasks on any accessible endpoint

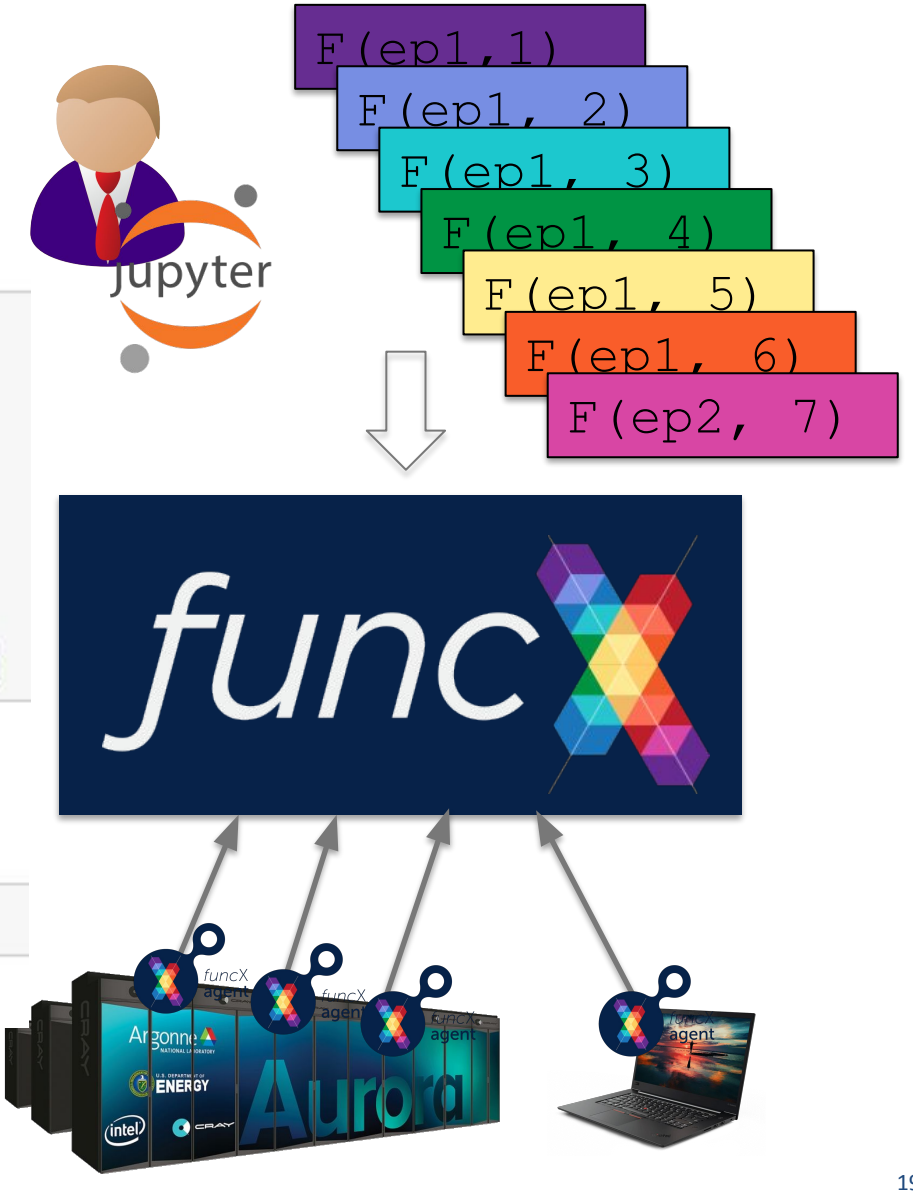
Choose a function, endpoint ID, and input arguments

```
from funcx.sdk.client import FuncXClient
from funcx.sdk.executor import FuncXExecutor

fx = FuncXExecutor(FuncXClient())

def hello():
    return 'Hello World'

future = fx.submit(hello, endpoint_id='4b116d3c-1703-4f8f-9f6f-39921e5864df')
```

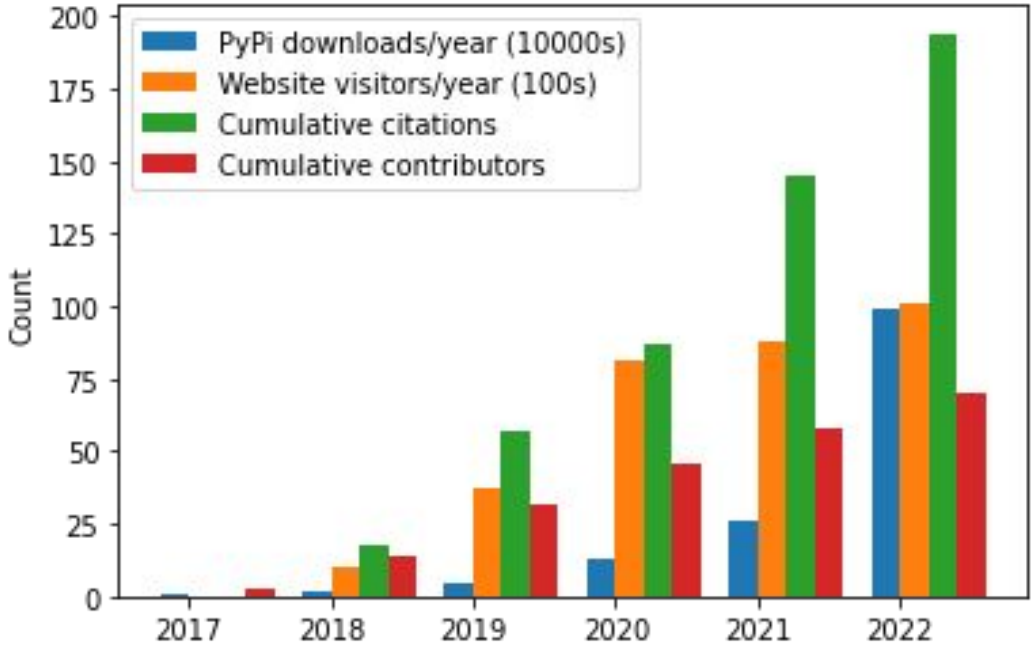
Asynchronously retrieve results

```
print(future.result())
```

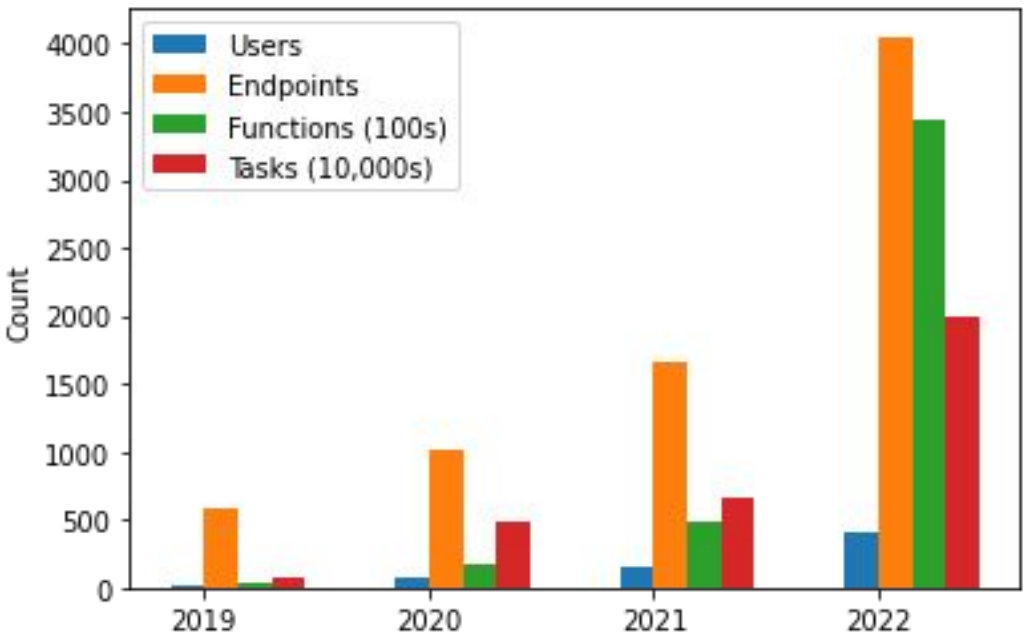
```
Hello World
```



Parsl and funcX continue to grow!



70 contributors, >1.4M PyPI downloads



>20M tasks, >340K functions, >4000 endpoints

Thank you funding agencies and project partners

2209919 (UChicago)
2209920 (UIUC)
1550588 (UChicago/UIUC)
1550476 (Notre Dame),
1550475 (Colorado State)
1550562 (Northern Arizona)
1550528 (College of New Jersey)

2004894 (UChicago)
2004932 (UIUC)

Argonne LDRDs

- 2022-0230 Productive Exascale Analysis Workflows for Numerical Cosmology
- 2021-0152 Creating a Robust and Scalable Framework for On-demand Analysis and AI-based Experiment Steering
- 2019-0217 Establishing a Usable, Scalable, and Reproducible Computational Ecosystem for Dark Energy Science

LSST Dark Energy Science Collaboration (DESC)

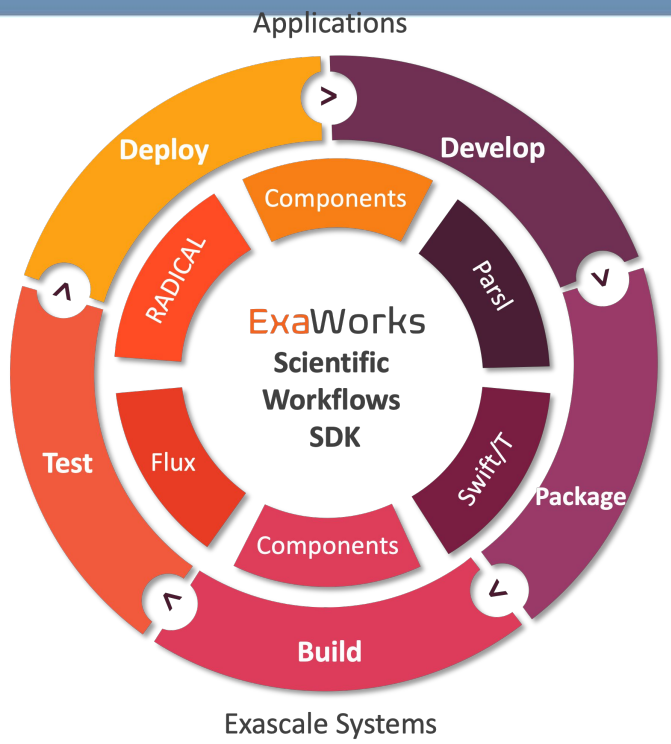
DOE ECP PRJ1008564 ExaWorks project

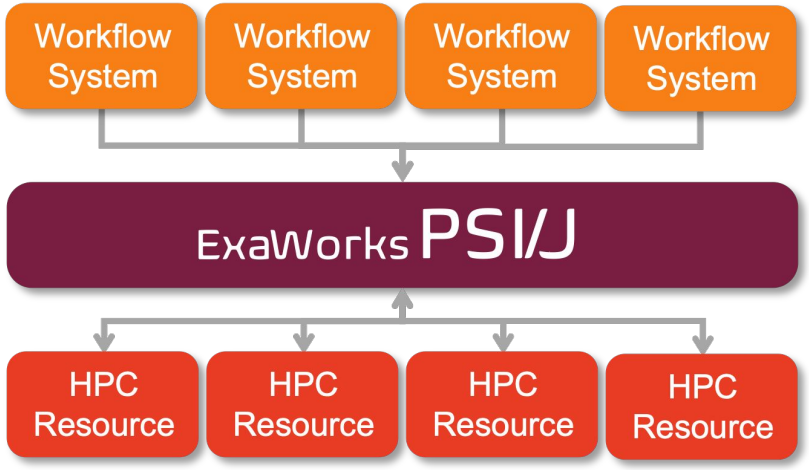
DOE DE-NA0003963 Center for Exascale-enabled Scramjet Design (CEESD)

Discovery Partners Institute (DPI): Airborne-Satellite-AI-HPC integrative framework (ASAI)

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)





workflows community initiative

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