A Brief Introduction to Parsl and funcX

Kyle Chard
chard@uchicago.edu

Parsl

Parsl: a parallel programming library for Python

Apps define opportunities for parallelism
- 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!

Parsl is an opensource community with 70+ contributors (https://parsl-project.org/parslfest.html)

```
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!
```

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

funcX: Fire-and-forget remote computing

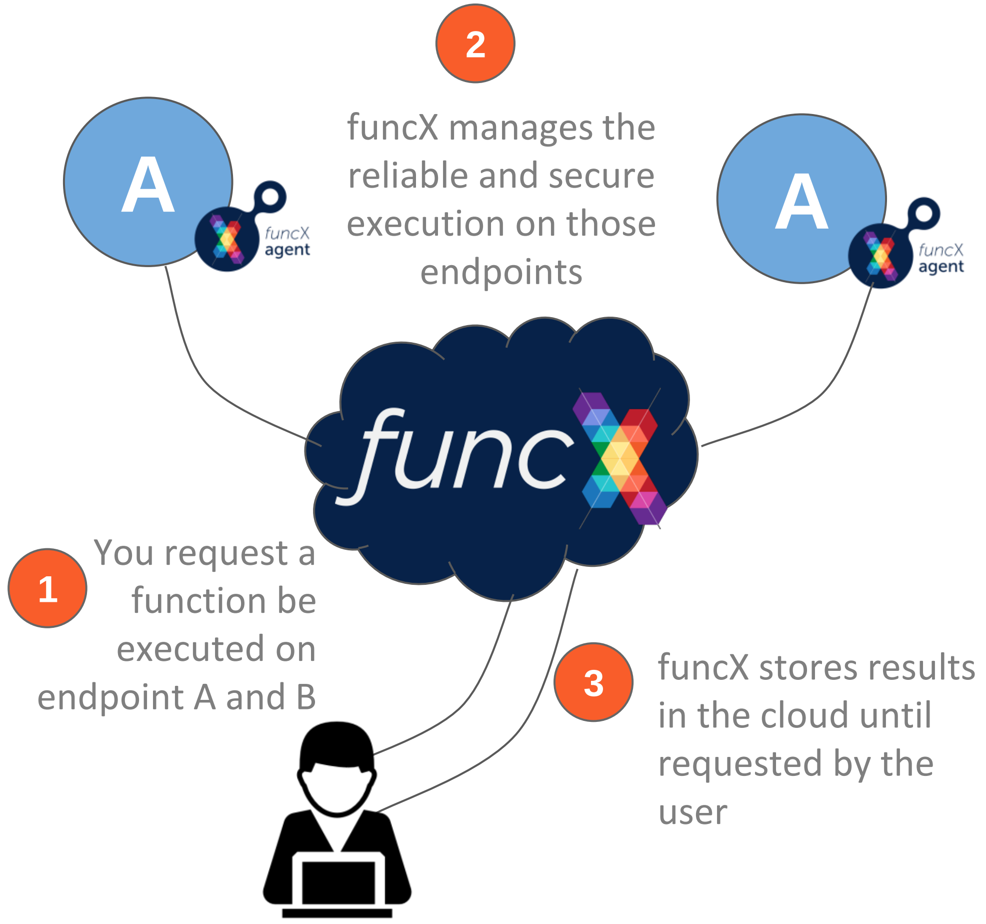
How do we coordinate work over several resources?

- Where do you persistently host the Parsl process? how do you reliably connect to remote resources? how do you recover from remote failures? …..

funcX servicifies Parsl:

- funcX endpoints (using Parsl) enable scalable execution of tasks on arbitrary resources
- funcX service provides robust fire-and-forget execution of tasks and asynchronous staging of results

funcX is developed by the Globus team and leverages the same security, deployment, operations model

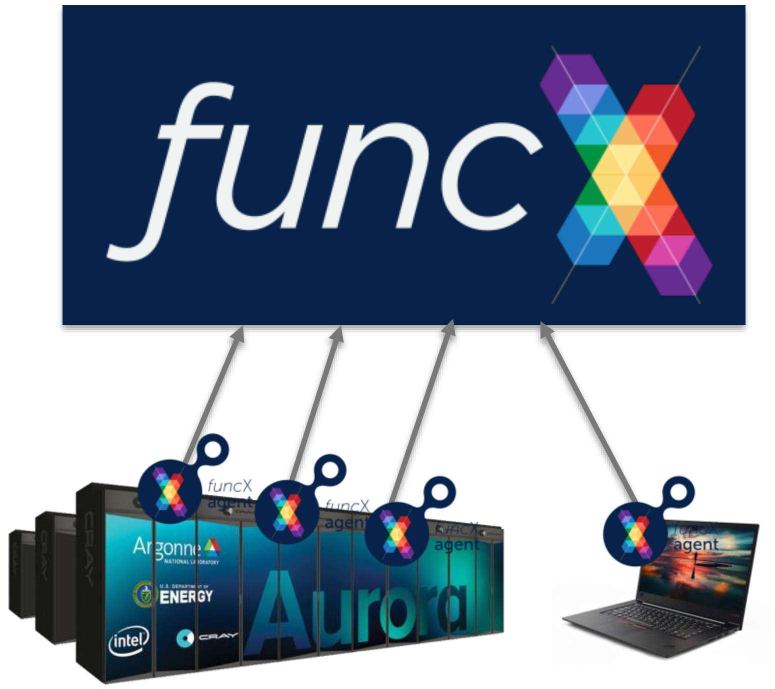

2
funcX manages the reliable and secure execution on those endpoints
A
funcX agent
A
funcX agent
funcX
1
You request a function be executed on endpoint A and B
3
funcX stores results in the cloud until requested by the user

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

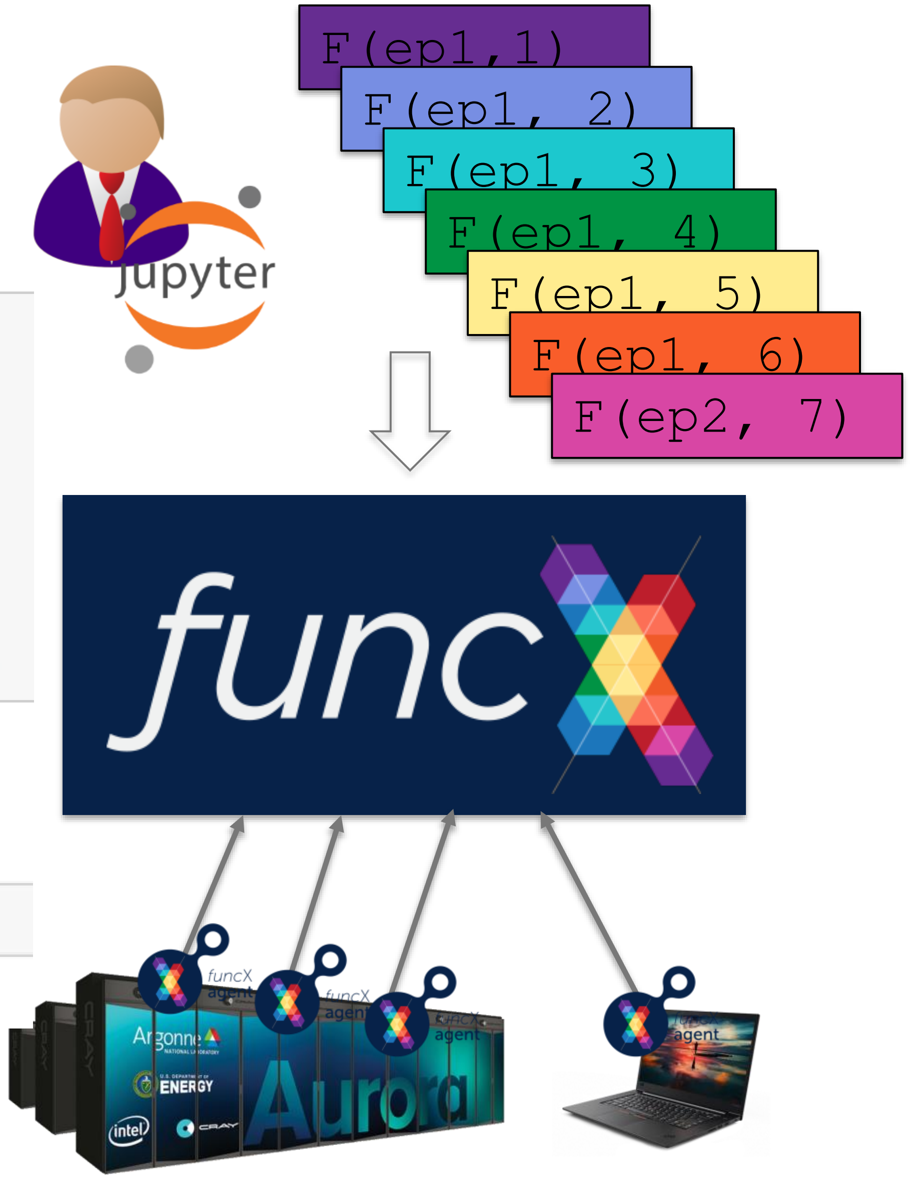
Choose a function, endpoint ID, and input arguments

```
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
```



Thank you funding agencies and project partners

Parsl

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

NSF

funcX

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)

U.S. DEPARTMENT OF ENERGY

Tutorial

```
$ module load conda
$ conda create --prefix ~/conda-envs/polaris-funcx python=3.9
$ conda activate polaris-funcx
```

Install and configure the funcX endpoint.

```
$ pip-install funcx-endpoint
$ funcx-endpoint configure polaris-endpoint
```

Optional: configure endpoint for Polaris queues

- https://funcx.readthedocs.io/en/latest/endpoints.html

Run Jupyter notebook (on Binder or your laptop)

- https://funcx.org/binder