Argonne NATIONAL LABORATORY

Workflows

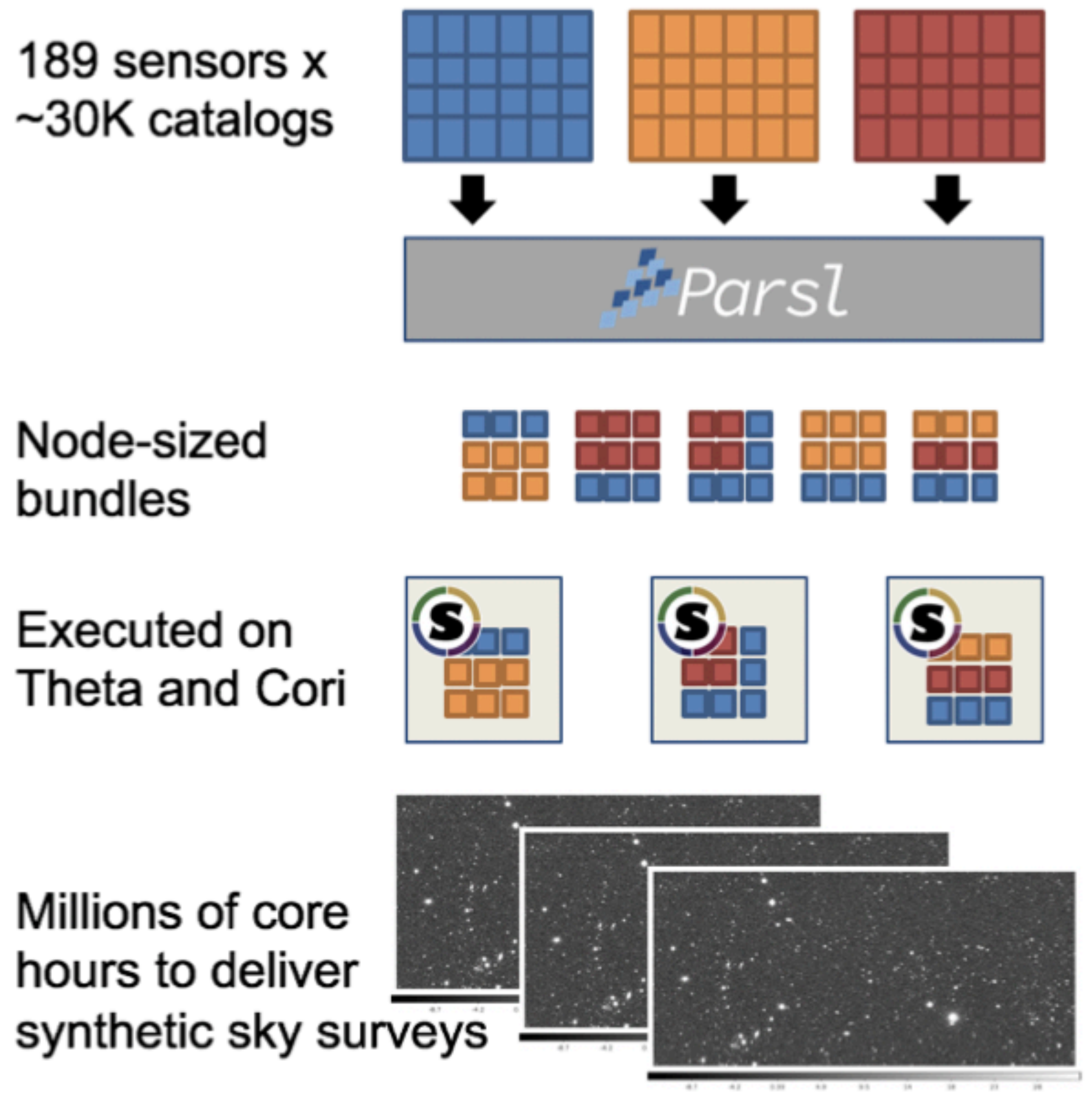
Christine Simpson
Assistant Computational Scientist
Data Services & Workflows Group, ALCF

ALCF Hands-on Workshop Sept 24, 2025

What is a workflow tool?

Why should you consider using one?

- A **workflow tool** is a piece of software that orchestrates the execution of large numbers of tasks on compute resources, handling dependencies, data flows, and errors/timeouts
- What a workflow tool can do for your workload:
    - **Run many tasks** concurrently and/or one after another asynchronously across one or many batch jobs
    - **Manage task dependencies**
    - **Automate** error handling and restarts of tasks
    - **Manage data movement** into/out of the file system needed for tasks
- ALCF and ANL have developed tools at the lab and in partnership with Globus Labs that run effectively on our machines


189 sensors x ~30K catalogs
Parsl
Node-sized bundles
Executed on Theta and Cori
Millions of core hours to deliver synthetic sky surveys

Villarreal et al. "Extreme Scale Survey Simulation with Python Workflows." Proceeding for eScience 2021

Workflow tools at ALCF

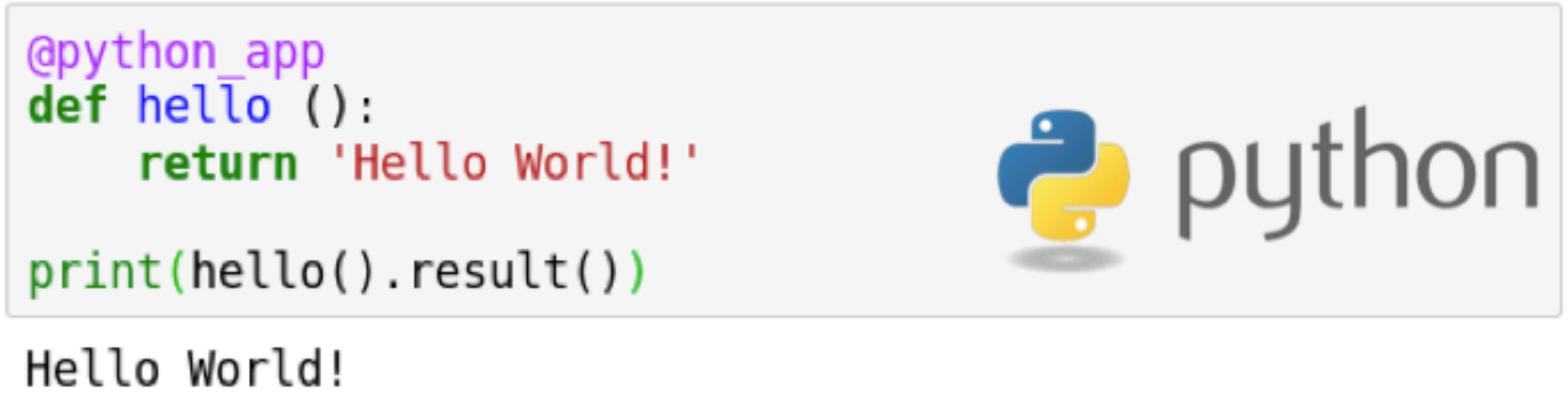
Parsl, Globus Compute/Flows, Dragon & Balsam

- Today, we will demo 3 tools commonly used at the facility for managing workflows
  - **Parsl** - developed by Globus Labs, UChicago and ANL; a good choice for locally executed, high throughput workflows executing tasks on single cores or nodes
  - **Globus Compute** - developed by Globus Labs; a good choice for remote execution of tasks
  - **Dragon** - developed by HPE; a distributed runtime that can manage tasks and in-memory data; has a python and C API
- There are many tools out there! I'll also briefly mention **Balsam**, an ALCF-developed tool that uses a database model. If you are interested in tools we don't cover today, please come talk to us and we can work with you

Parsl

A parallel programming library for Python

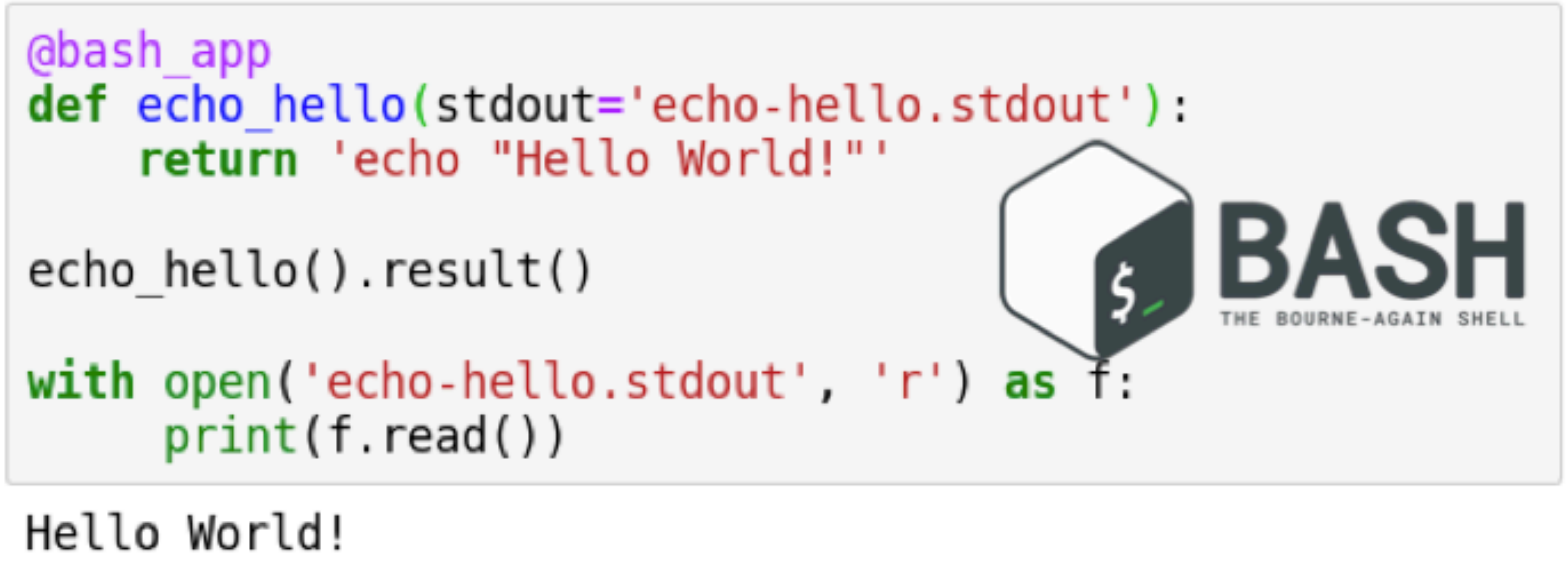
- Simple installation with pip
- Apps define how to run tasks
  - Python apps call python functions
  - Bash apps call external applications
- Workflow contained within memory (no database)
- Configuration (assignment of tasks to hardware) set by user, separate from workflow logic and application definitions
- Apps return futures: a proxy for a result that might not yet be available
- Apps run concurrently, respecting dependencies
- Community of 70+ developers, several at UChicago & ANL, part of Globus Labs



```
@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!
```

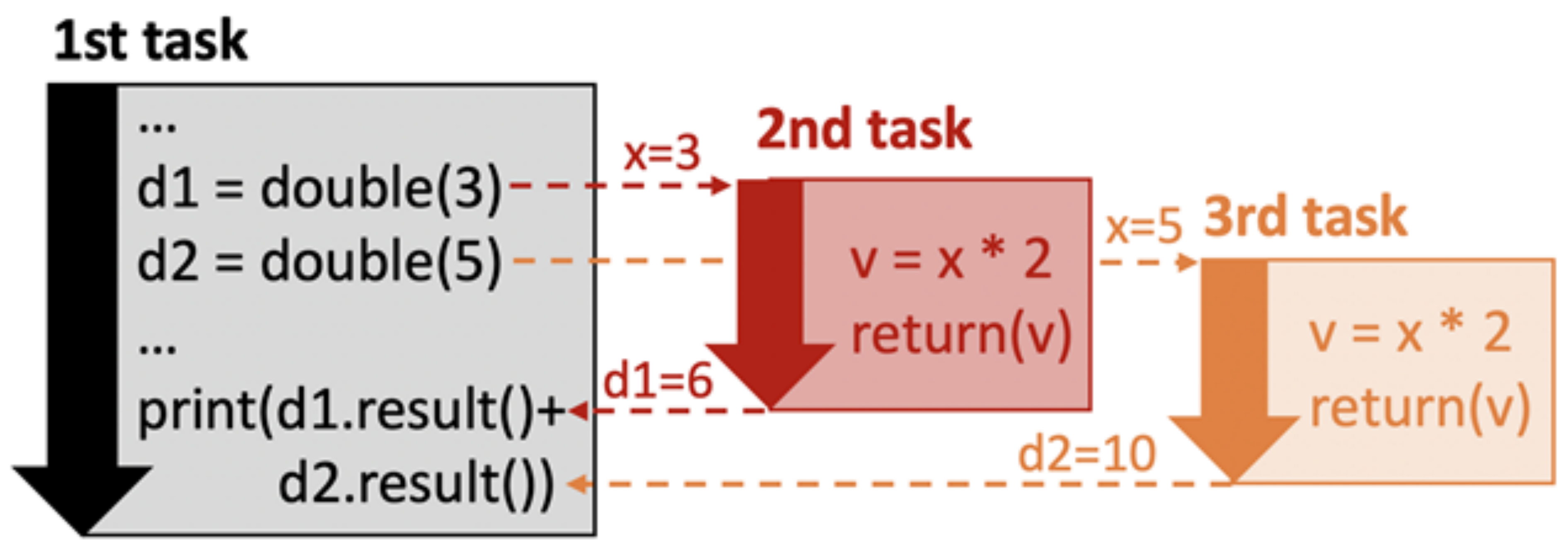
`Parsl` Apps and Futures

How tasks are made and linked

- Parsl extends the Python `concurrent.futures` module
- Tasks are created by invoking apps that return an AppFuture
- Task dependencies can be created by passing the AppFuture from one task to another

Concurrent Tasks

```
@python_app
def double(x):
  v = x * 2
  return v
```



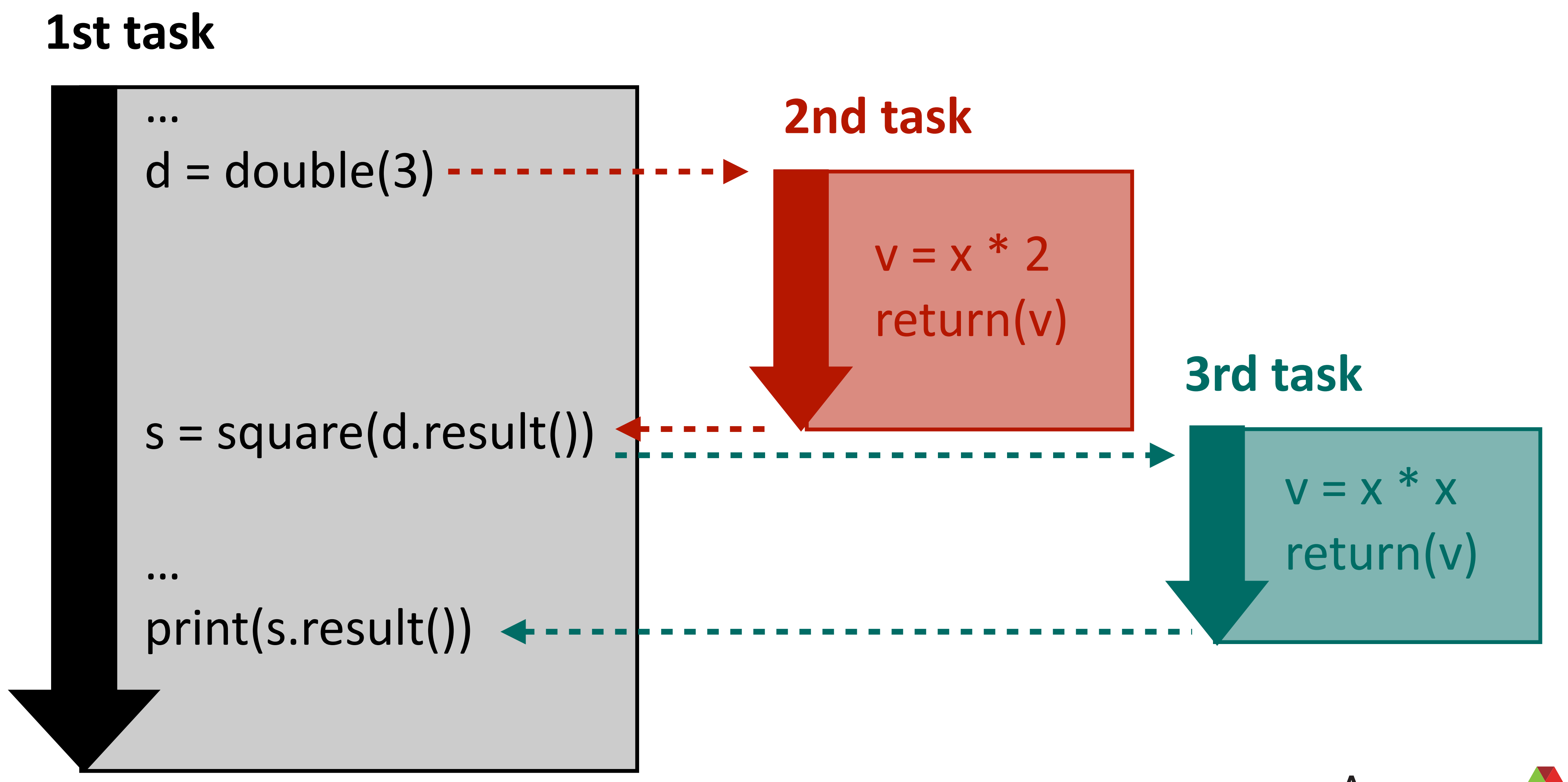
```
...
d1 = double(3)
d2 = double(5)
...
print(d1.result()+
      d2.result())
```

```
v = x * 2
return(v)
```

Dependent Tasks

```
@python_app
def double(x):
  v = x * 2
  return v

@python_app
def square(x):
  v = x * x
  return v
```



```
...
d = double(3)

s = square(d.result())

...
print(s.result())
```

```
v = x * 2
return(v)
```

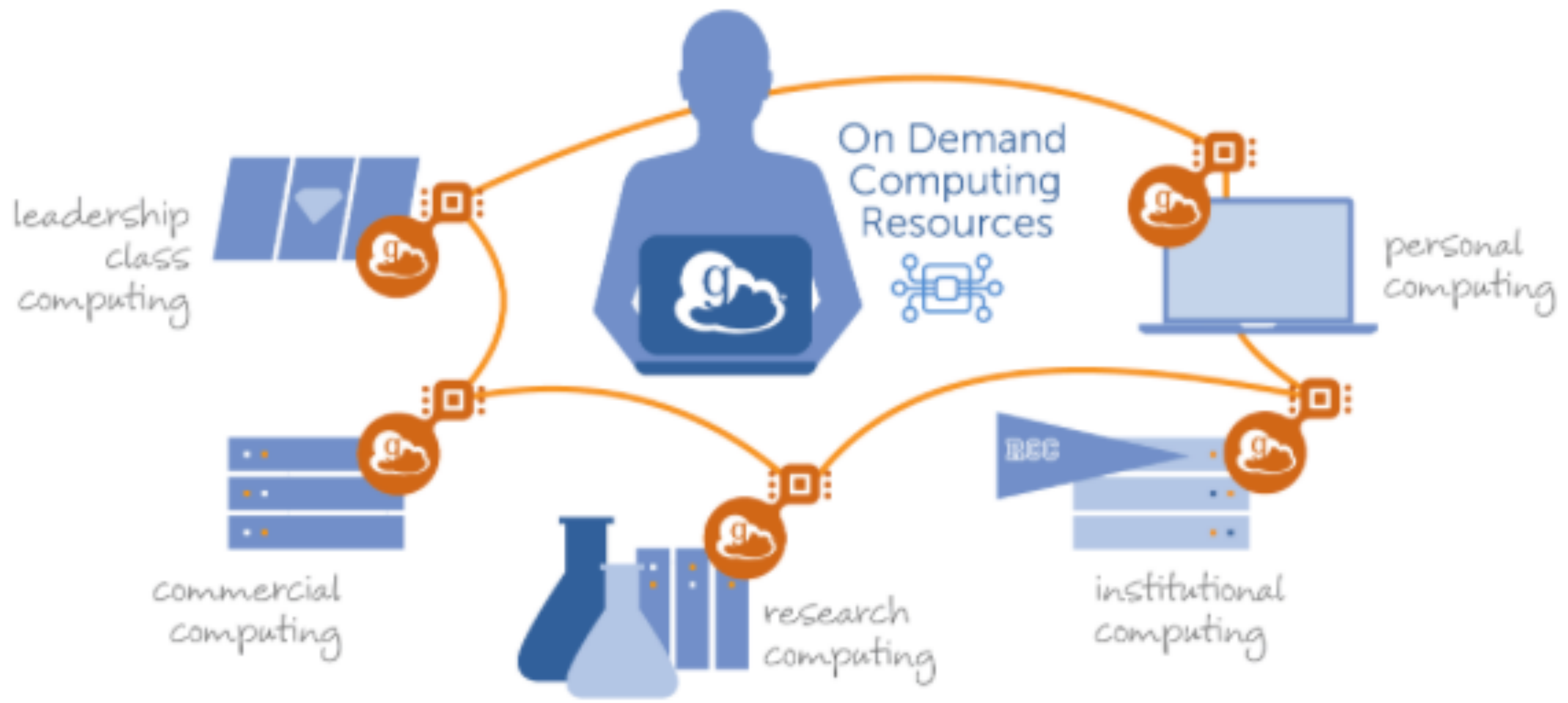
```
v = x * x
return(v)
```

Parsl Demo

Globus Compute

“fire-and-forget” execution of tasks

- Allows users to launch applications remotely from laptop, other machine, etc.
- Built on top of Parsl, similar configuration, also uses python futures
- Allows users to launch applications remotely from laptop, external machine, anywhere
- Requires the setup of a Compute Endpoint on the target machine (e.g. Polaris) beforehand
- Globus Compute functions can be integrated with data transfers with Globus Flows



Globus Compute Demo

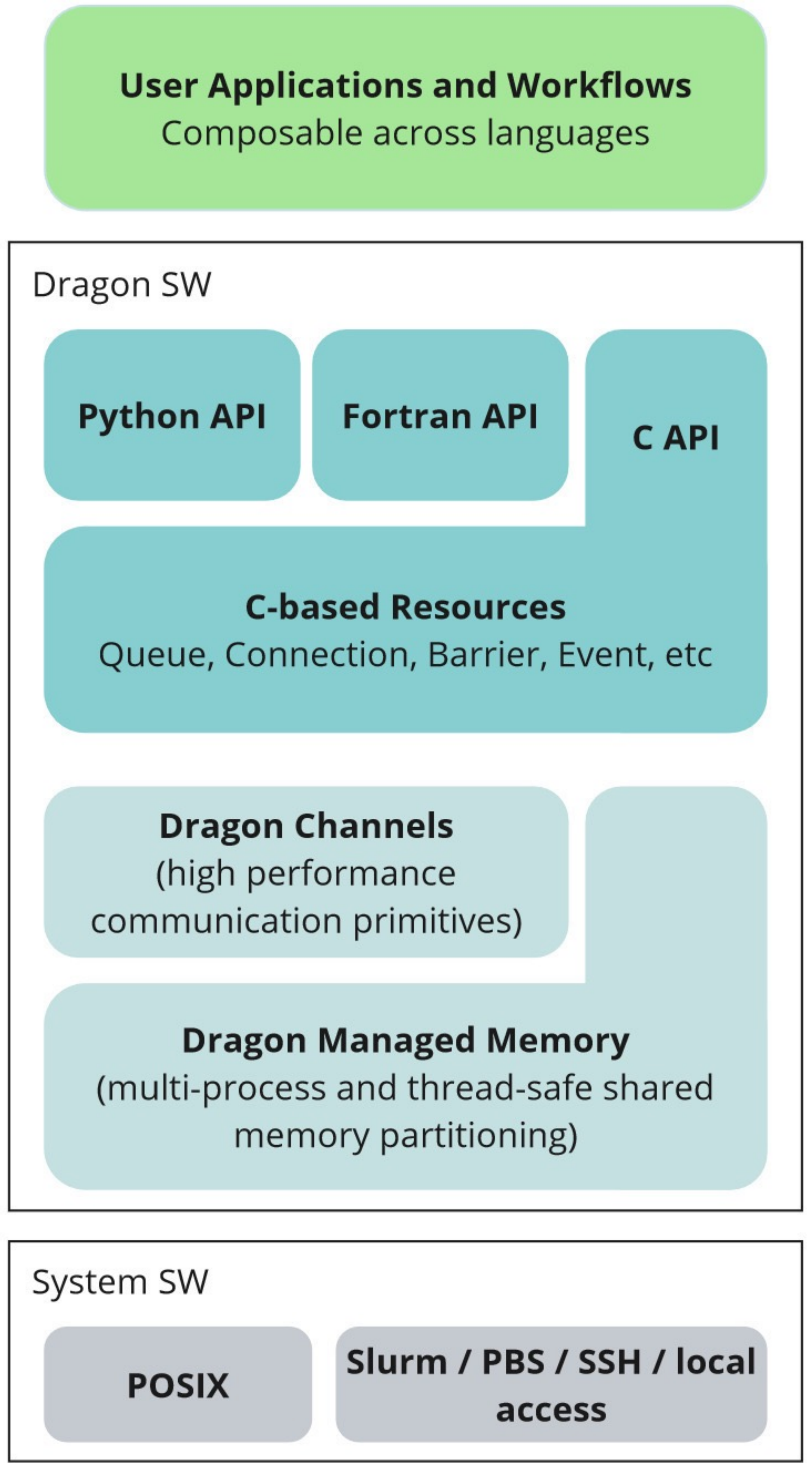
Dragon HPC

DragonHPC

Hewlett Packard Enterprise

Distributed runtime for tasks and data movement

- Open source project developed by HPE
- Python API is multi-node extension to Python multiprocessing (e.g. mp.Process, mp.Pool, …)
- C API included, Fortran API in development
- Managed memory through sharded dictionary objects
- Parallel process launching with fine-grained control of CPU/GPU affinity
- High-speed RDMA transport agents for off-node communication on Slingshot and Infiniband networks (TCP for other networks)
- Interfaces for higher-level workflow tools, e.g. SmartSim (and in development for Parsl and Dask)
- Install with pip


User Applications and Workflows
Composable across languages
Dragon SW
Python API
Fortran API
C API
C-based Resources
Queue, Connection, Barrier, Event, etc
Dragon Channels
(high performance communication primitives)
Dragon Managed Memory
(multi-process and thread-safe shared memory partitioning)
System SW
POSIX
Slurm / PBS / SSH / local access

Dragon Demo

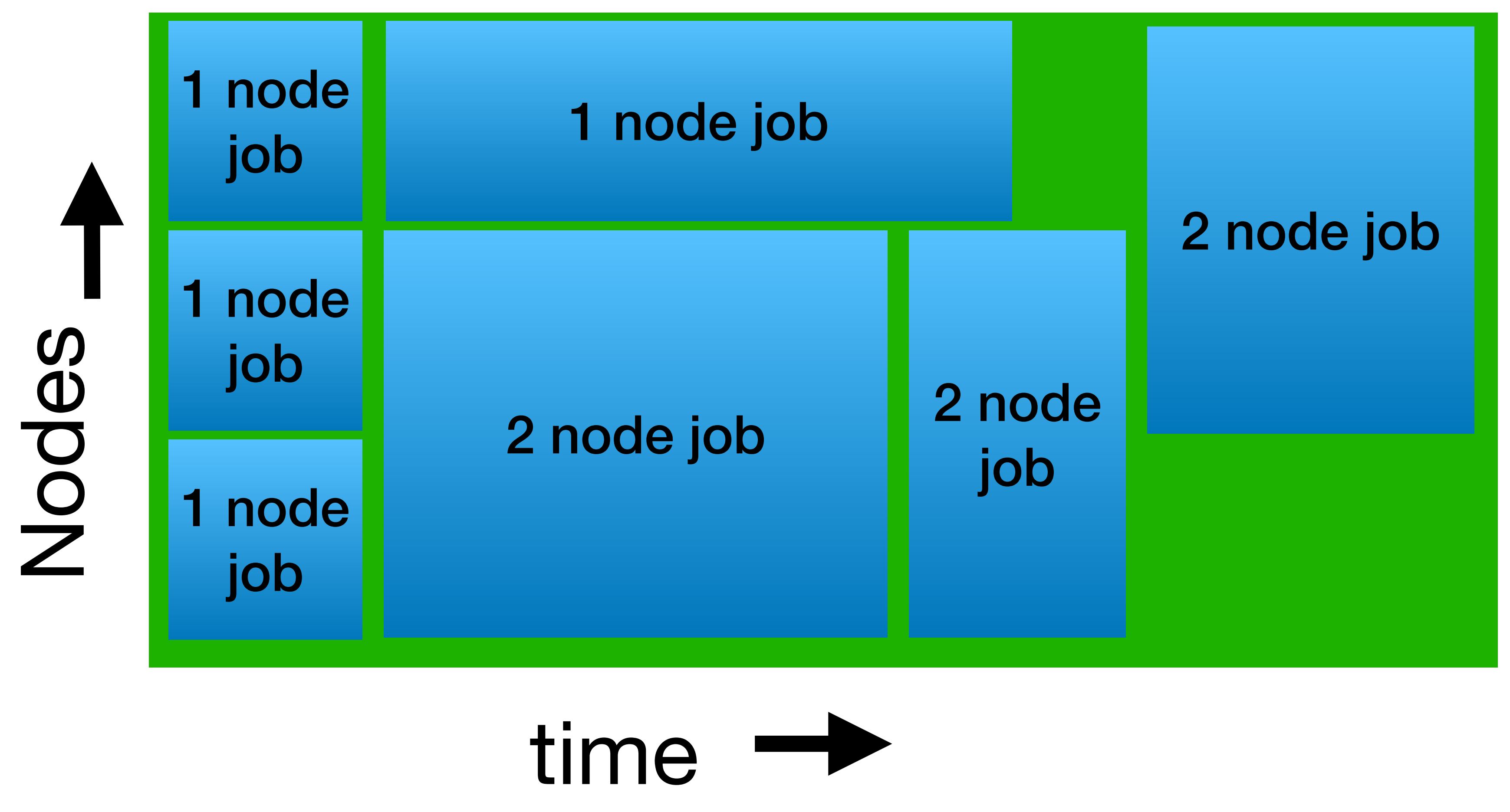
Balsam Workflow Management Tool

A unified platform to manage high-throughput workflows across the HPC landscape

- Balsam was developed at ALCF and is used for deploying workflows on DOE HPC machines
- Balsam uses a **database model**, applications and tasks are stored in a centralized database that tracks the progress of tasks, called jobs
- Install with pip, has a **Python API** and **command line interface**
- Can execute external apps and native python apps
- Optimized for running **MPI** applications
- Centralized server allows for inter-machine workflows
- Database hosted for the user at ALCF (requires ALCF account)
- Supported configurations for ALCF machines, and machines at NERSC & OLCF

To use Balsam, request access to Balsam server by email: support@alcf.anl.gov or drop a request in the #technical-q-a channel

3 Node Batch Job running 7 Balsam jobs requiring different run times and node numbers



Balsam Apps and Jobs

How to manage work

Define applications as Python classes, e.g.:

```
from balsam.api import ApplicationDefinition, Job, BatchJob

class Lammps(ApplicationDefinition):

    site = “polaris_tutorial"

    def shell_preamble(self):
        return f’export PATH=/path/to/lmp:$PATH’

    command_template = 'lmp –in /path/to/input.in –var tinit 
{{tinit}}'

Lammps.sync()
```

Query, track, and execute Jobs from the command line (or through python API), e.g.:

```
> balsam job ls
ID          Site                App       Workdir       State           Tags
34017534    polaris_tutorial    Lammps    lat_1/run0    PREPROCESSED    {'case': 'lattice_1'}
34017535    polaris_tutorial    Lammps    lat_1/run1    PREPROCESSED    {'case': 'lattice_1'}
34017536    polaris_tutorial    Lammps    lat_2/run0    JOB_FINISHED    {'case': 'lattice_2'}
34017537    polaris_tutorial    Lammps    lat_2/run1    JOB_FINISHED    {'case': 'lattice_2'}
34017538    polaris_tutorial    Vasp      vasp/test0    PREPROCESSED    {'compound': 'test'}
34017539    polaris_tutorial    Vasp      vasp/test1    PREPROCESSED    {'compound': 'test'}
```

More Resources

- **Parsl**
  - docs: https://parsl.readthedocs.io/en/stable/
  - github: https://github.com/Parsl/parsl
  - slack: https://parsl-project.org/support.html
- **Globus Compute**
  - docs: https://globus-compute.readthedocs.io/en/latest/quickstart.html
  - slack: https://join.slack.com/t/funcx/shared_invite/zt-3ehs7wjm8-wtwHUjzm3YAvZ20Pmh9tbA
- **Dragon**
  - https://dragonhpc.org/portal/index.html
  - https://dragonhpc.slack.com/
  - GitHub: https://github.com/DragonHPC/dragon
- **Balsam**
  - docs: https://argonne-lcf.github.io/balsam/
  - github: https://github.com/argonne-lcf/balsam
  - slack: https://join.slack.com/t/balsam-workflows/shared_invite/zt-1t0736hsz-6hxsmC~0MBFpuP~WvouwWQ
- Workflows workshop materials (includes materials on how to run GNU Parallel, Parsl, Balsam & Fireworks on Polaris): https://github.com/CrossFacilityWorkflows/DOE-HPC-workflow-training
- Workflows community (group where you can discover new workflow tools & connect with workflows community) : https://workflows.community/

Thank-you!

Questions?