ParslFest 2020

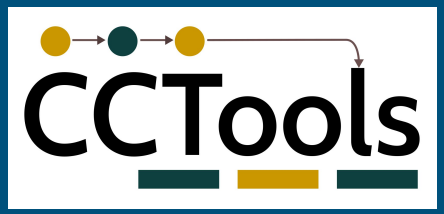
Fine-grain management of resources with WorkQueue

Tim Shaffer
tshaffe1@nd.edu

Zhuozhao Li
zhuozhao@uchicago.edu

Ben Tovar
btovar@nd.edu

UNIVERSITY OF NOTRE DAME



Work Queue Executor

Using WQ with Parsl

Work Queue is a manager-worker framework for executing tasks on a pool of workers

- Similar use cases as HighThroughputExecutor
  - Pilot job model allows many small tasks to run without waiting in the batch queue
  - Pack multiple tasks per worker node

- Plus some additional features
  - WQ handles file transfers by default, so no shared FS required
  - Workers cache common input files, reducing transfer times
  - **Fine-grained resource management**
  - **Automatic dependency management**

Install via Conda

Make sure Conda is installed and set up first

```
# create and activate a Conda environment
$ conda create -y --name <environment>                \
    python=<version> pip
$ conda activate <environment>

# install CCTools and Parsl
$ conda install -c conda-forge ndcctools
$ pip install parsl
```

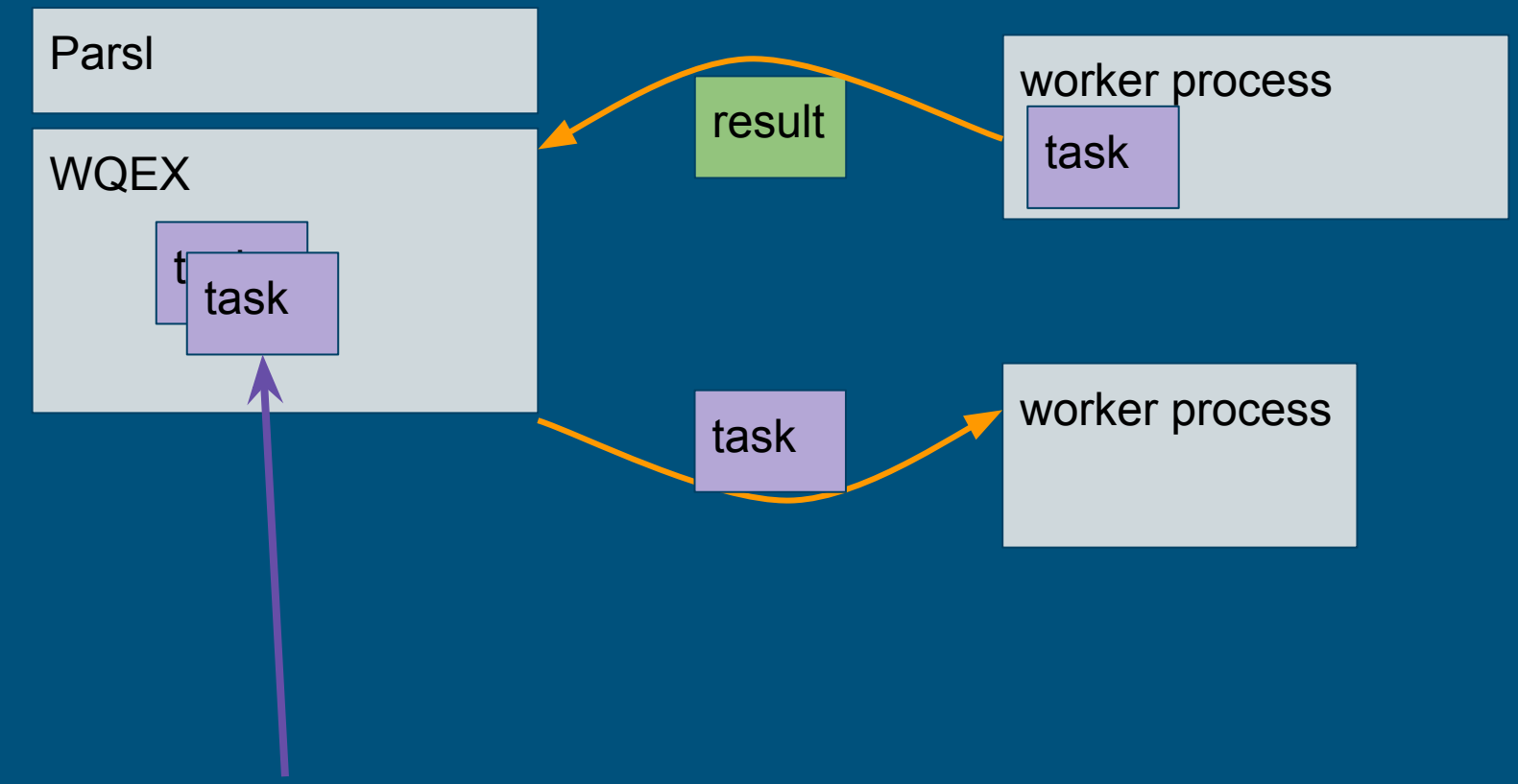
Starting Workers

Factory creates workers as needed:

```
$ work_queue_factory -Tcondor \
 -M my-app
 --min-workers 5
 --max-workers 200
 --cores 1 --memory 4096 --disk 10000
 --tasks-per-worker 4
```

Many batch systems supported:
SGE, Slurm, Condor, Torque, AWS Lambda, ...

Parsl + WQ


Parsl
WQEX
task
result
worker process
task
task
worker process
queue of tasks to be done

Fine-grained Resource Management

Resources Contract:
running several tasks in a worker concurrently

```
Worker has
available:

i cores
j MB of memory
k MB of disk
```

```
Task needs:



m cores
n MB of memory
o MB of disk
```

Task runs only if it fits in the currently available worker resources.

Resources Contract example

```
Worker has
available:

8 cores
512 MB of memory
512 MB of disk
```

```
Task a:

4 cores
100 MB of memory
100 MB of disk
```

```
Task b:

3 cores
100 MB of memory
100 MB of disk
```

Tasks a and b may run in worker at the same time.
(Work could still run another 1 core task.)

Managing Resources

Do nothing (default if tasks don't declare cores, memory or disk):

One task per worker, task occupies the whole worker.

Honor contract (default if tasks declare resources):

Task declares cores, memory, and disk (all three of them!)

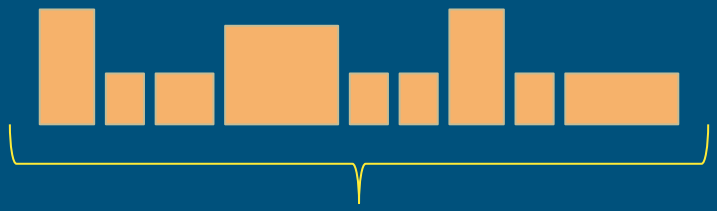
Worker runs as many concurrent tasks as will fit.

Tasks **may** use more resources than declared.

Automatic resource labeling:

Tasks are retried with resources that maximize throughput.

Automatic Resource Labeling: When you don't know how big your tasks are

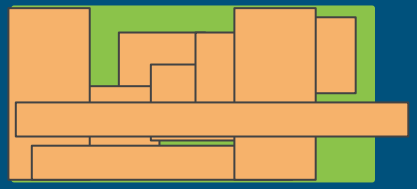


Tasks whose size
(e.g., cores, memory, and disk)
is not known until runtime.

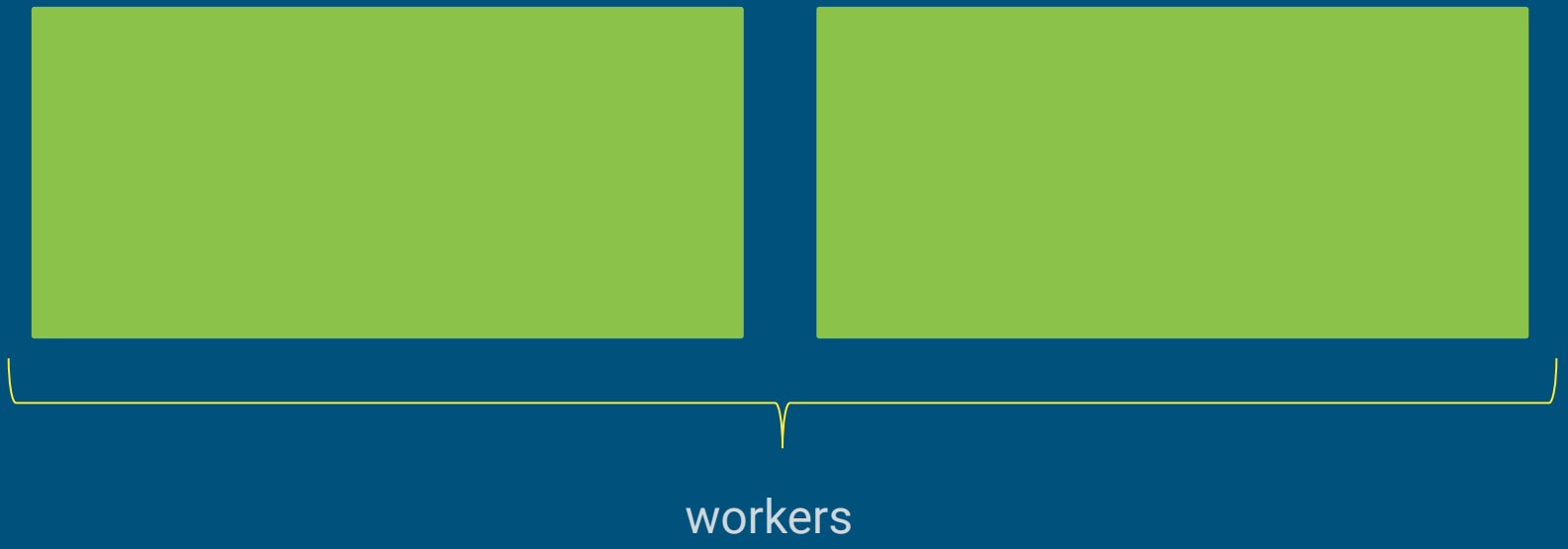
workers

One task per worker:
Wasted resources, reduced throughput.

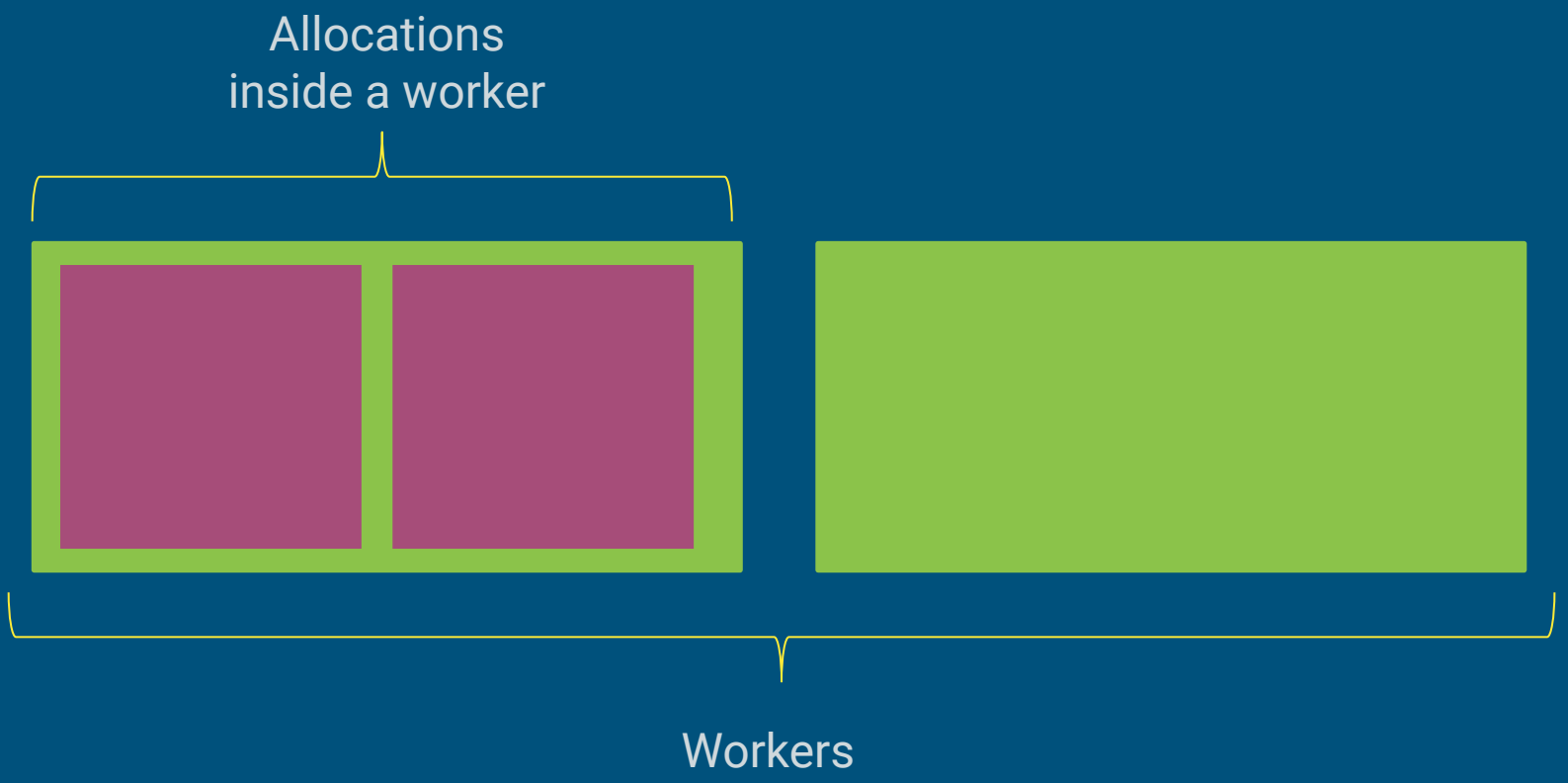
Many tasks per worker:
Resource contention/exhaustion,
reduced throughput



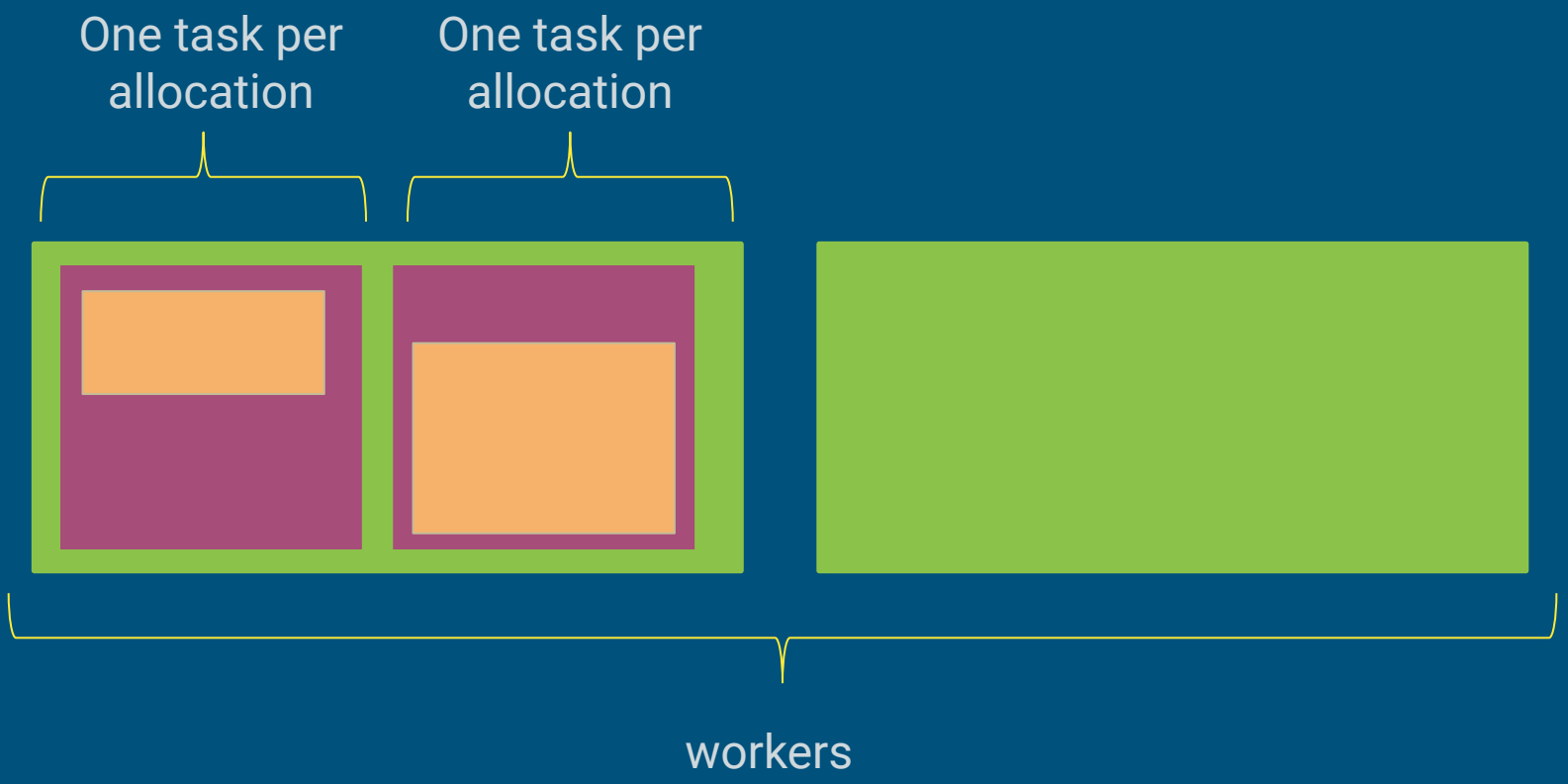
Task-in-the-Box



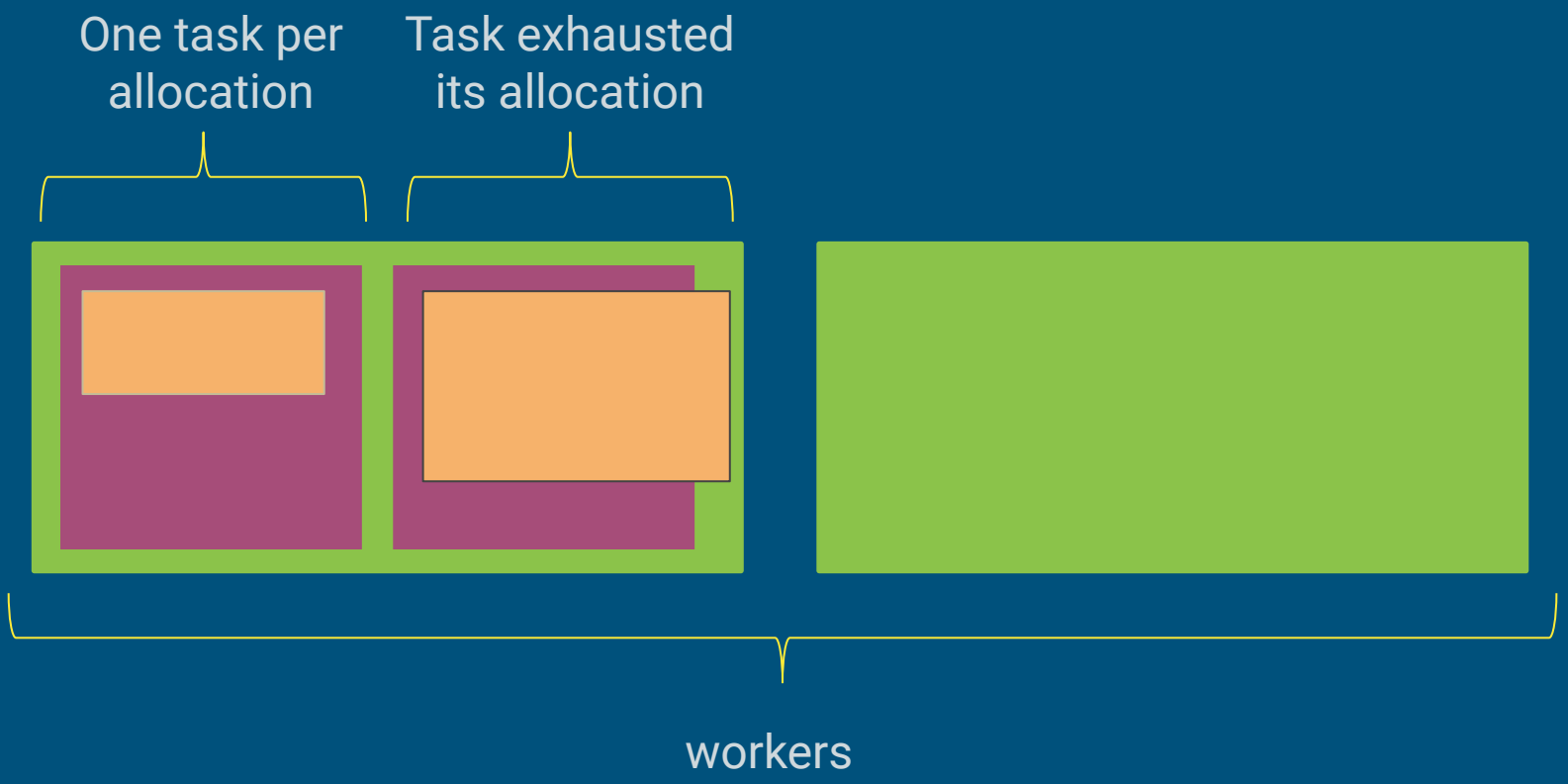
Task-in-the-Box



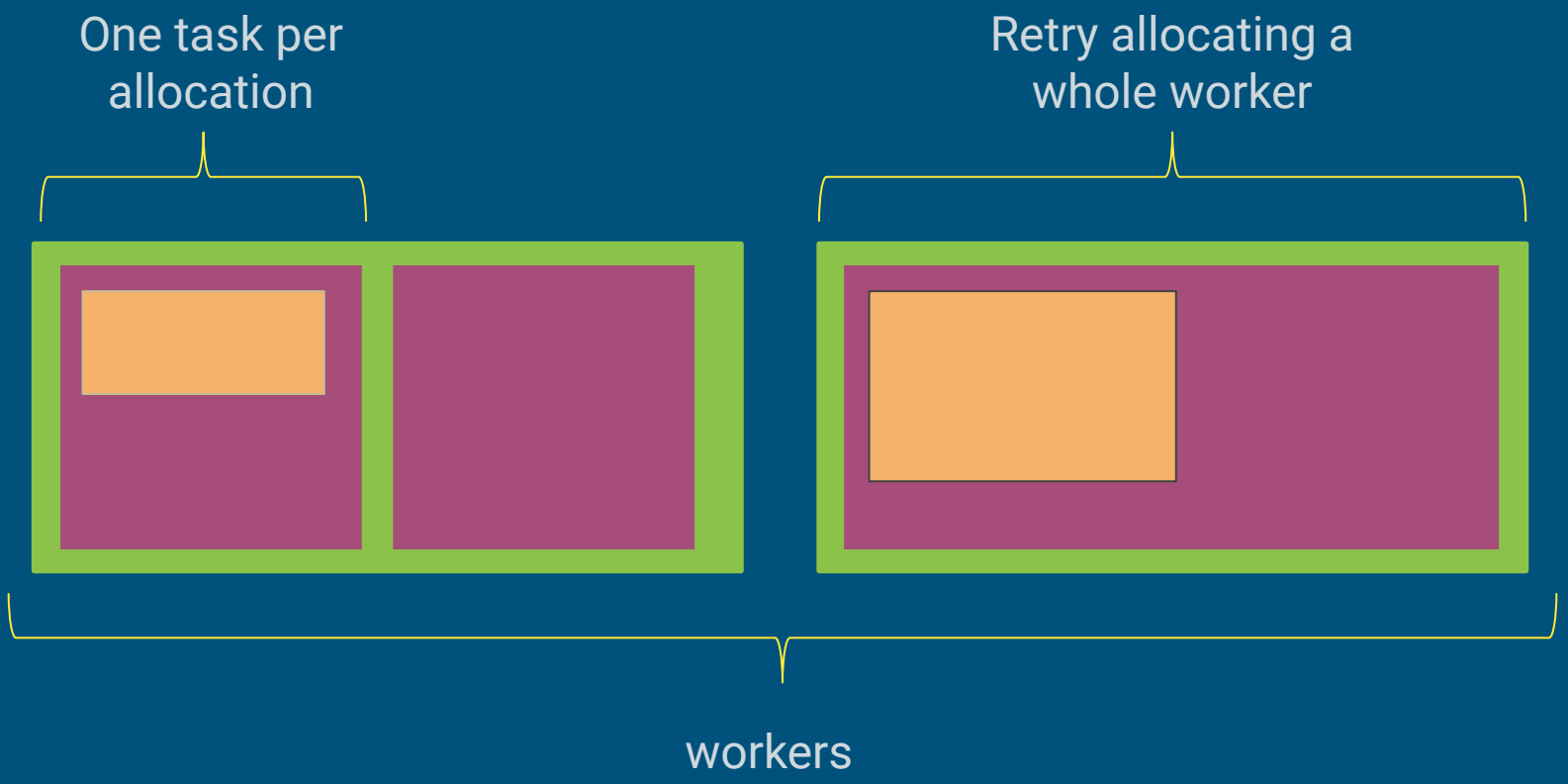
Task-in-the-Box



Task-in-the-Box



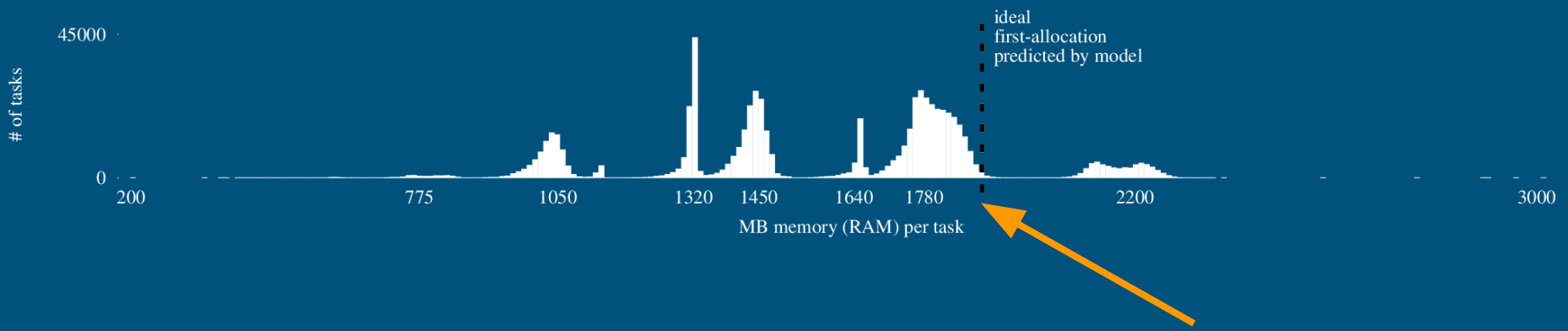
Task-in-the-Box



ND CMS example

Real result from a production High-Energy Physics CMS analysis
(Lobster NDCMS)

Histogram showing Peak Memory vs Number of Tasks
O(700K) tasks that ran in O(26K) cores managed by WorkQueue/Condor.

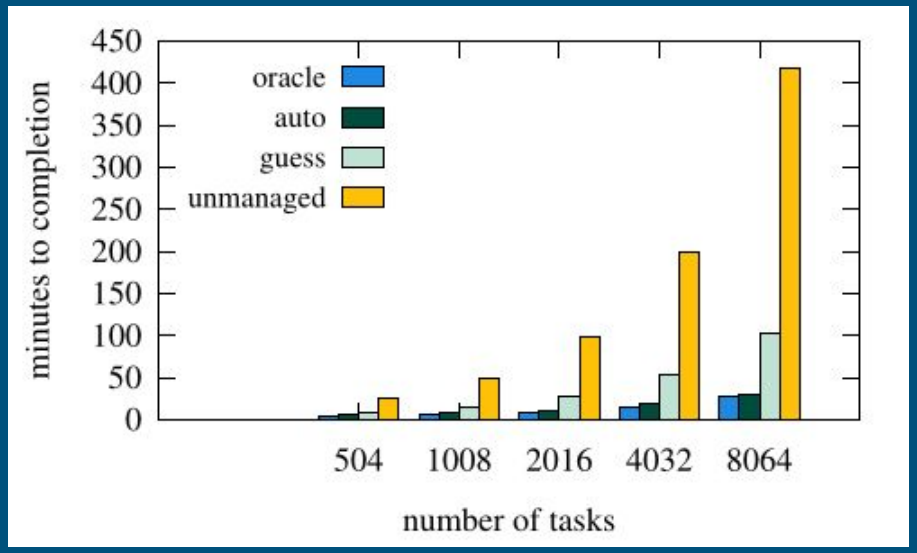


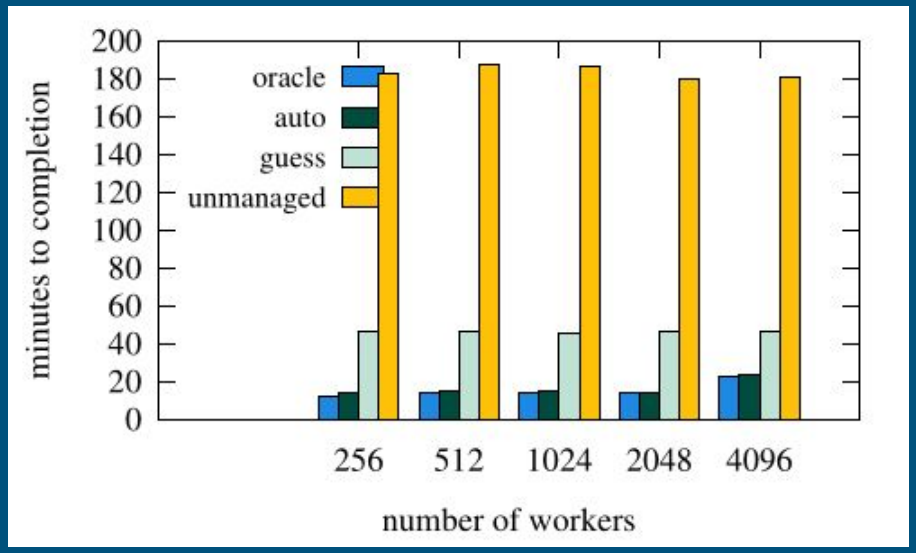
First-allocation that maximizes expected throughput
(increase of %40 w.r.t. no task is retried)

Tovar, et.al
DOI:10.1109/TPDS.2017.2762310

Scaling example (CANDLE)

oracle: exact resource requirements
auto: WQ's autolabeling
guess: reasonable static guess
unmanaged: task consumes whole worker





Automatic Dependency Management

Dependencies in Parsl Apps

Apps must explicitly import dependencies

```
@python_app
def do_something(x):
    import numpy
    y = numpy.linspace(0, 3, 100)
    return numpy.sin(x + y)
```

```
# But when the task runs on workers....
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'numpy'
```

Dependencies in Parsl Apps

A shared FS can mask dependency issues: transfers happen in the background without Parsl's involvement

- Some batch systems (e.g. Condor) may lack shared FS support
- Can't use shared FS across multiple sites
- Imports are a hidden cost:
  - Ever have to wait while workers `import tensorflow` ?
  - Shared FS performance can get worse at scale

Shaffer, et.al
DOI: 10.1145/3149393.3149401

Dependency management with CCTools

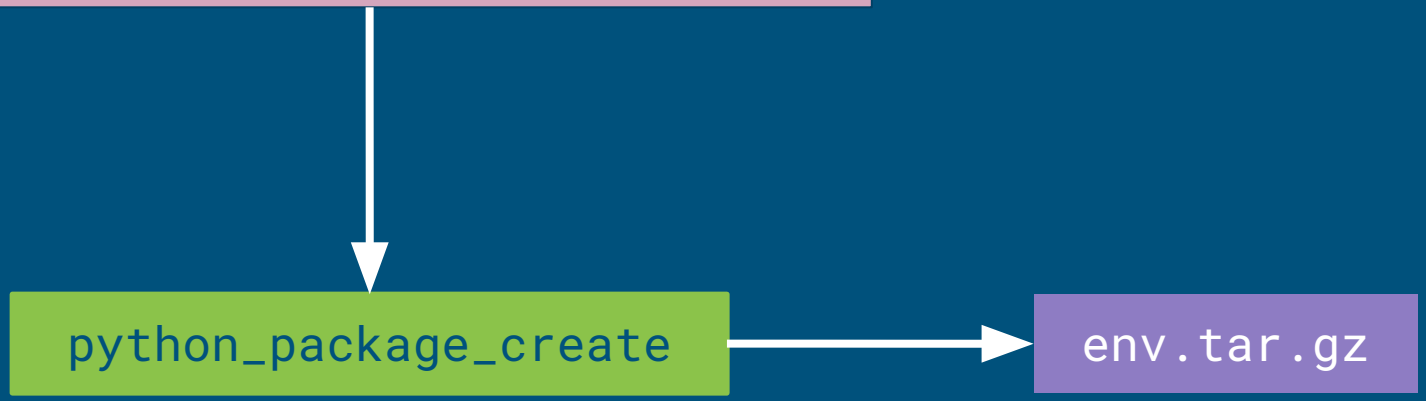
```python
@python_app
def do_something(x):
    import numpy
    y = numpy.linspace(0, 3, 100)
    return numpy.sin(x + y)
```

python_package_analyze

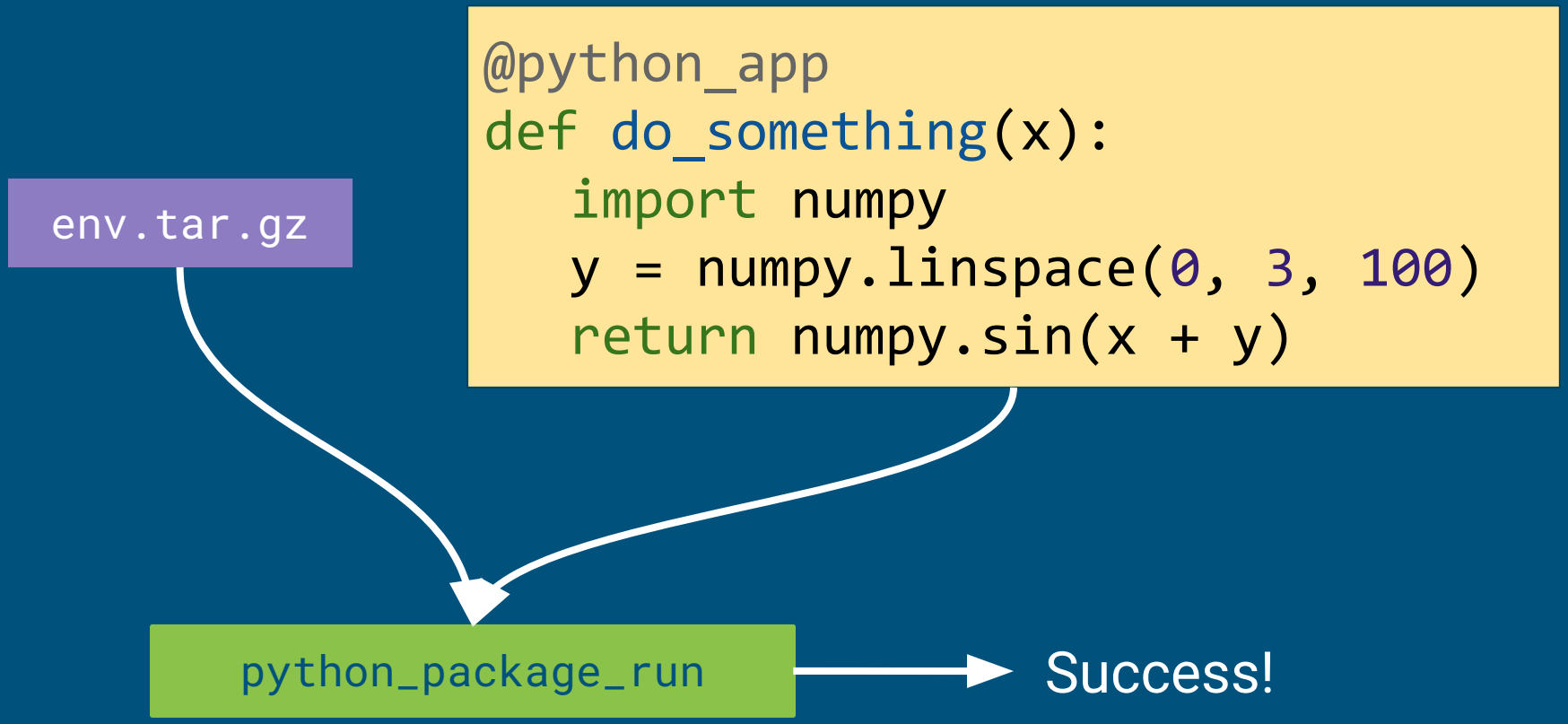
```
{
  "channels": [
    "conda-forge",
    "defaults"
  ],
  "dependencies": [
    "python=3.7.6=cpython_h8356626_6",
    "numpy=1.19.1=py37h7ea13bd_2",
...
```

Dependency management with CCTools

```
{
  "channels": [
    "conda-forge",
    "defaults"
  ],
  "dependencies": [
    "python=3.7.6=cpython_h8356626_6",
    "numpy=1.19.1=py37h7ea13bd_2",
...
```



Dependency management with CCTools



Automatic Dependency Management

The Work Queue Executor can handle these steps automatically (remember this a beta feature, might need some tinkering to get going)

Packages also include Python itself, so this works even if Python is unavailable/wrong version on workers!

Works well with WQ's built-in caching

Configuring the Work Queue Executor

`autolabel=True`

Use WQ's resource monitoring to infer task requirements

`autocategory=True`

Track and label each App separately

`pack=True`

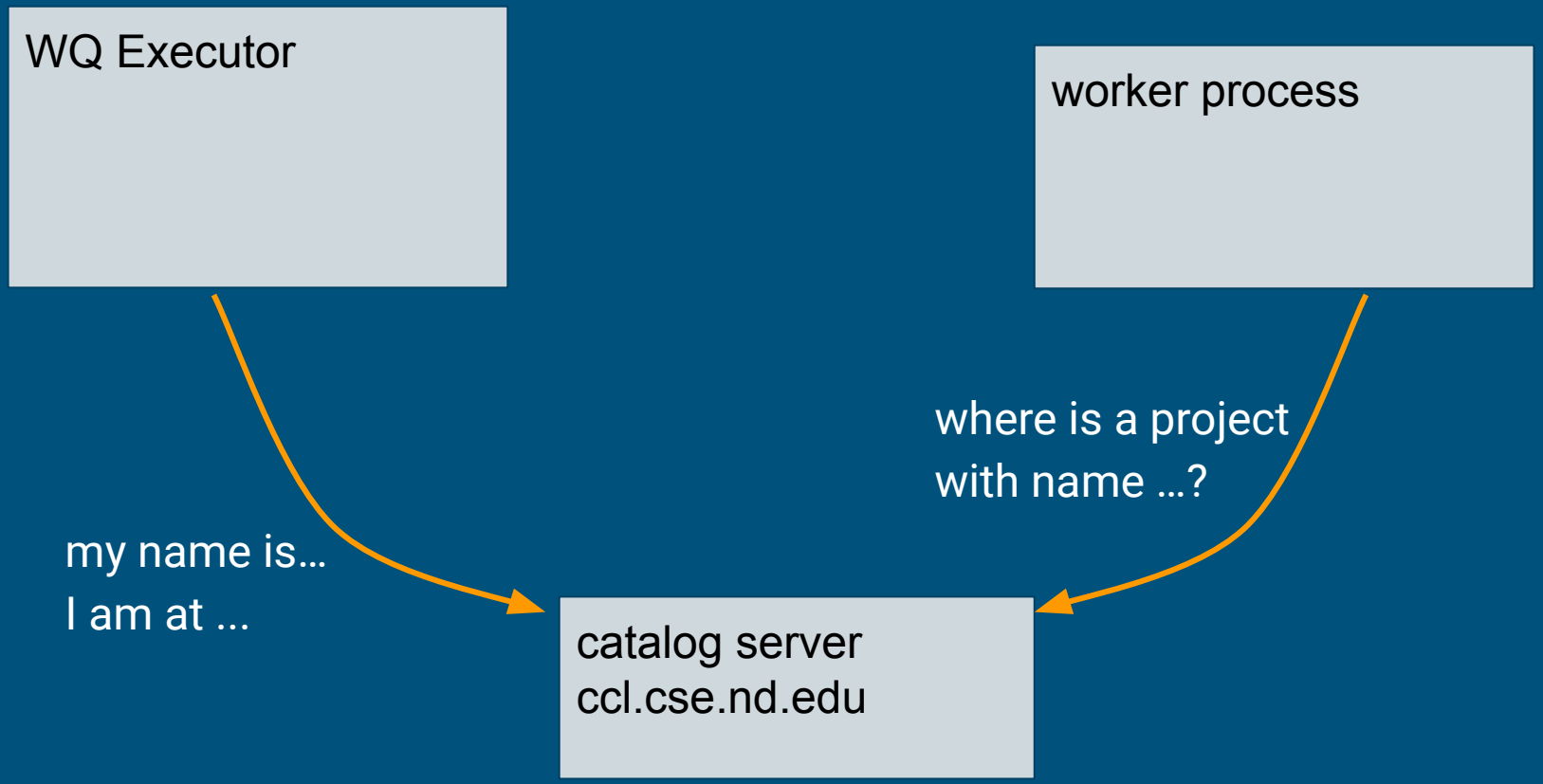
Prepare packaged environments for Python Apps

Demo

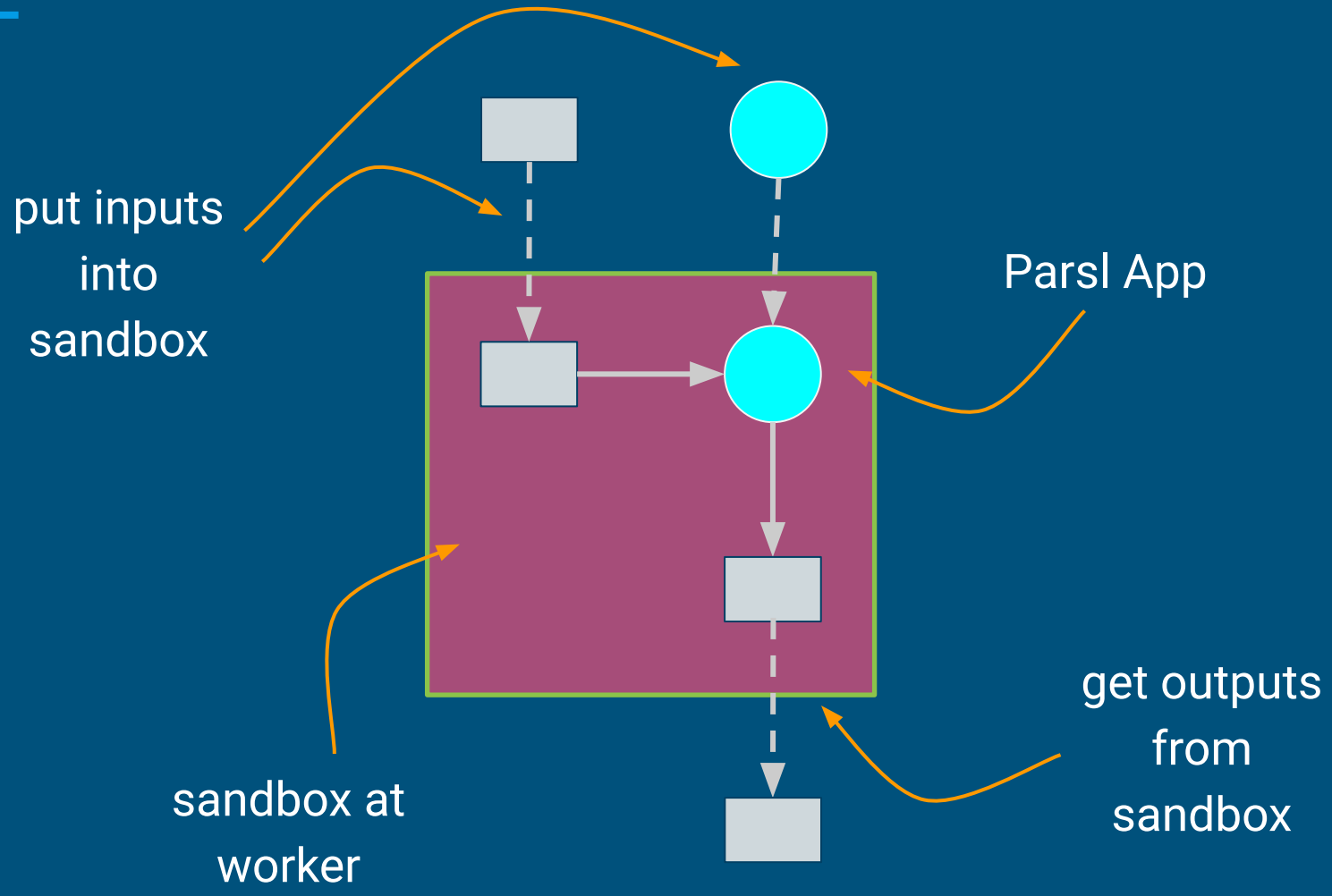
Test your setup

```
# if the following command fails, check your Conda env
$ work_queue_worker --version
work_queue_worker version  7.0.13 FINAL from source (released 2019-05-14 09:42:11 -0400)
     Built by btovar@camd04.crc.nd.edu on 2019-05-14 09:42:11 -0400
     System: Linux camd04.crc.nd.edu 3.10.0-957.el7.x86_64 #1 SMP Thu Oct 4 20:48:51 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux
     Configuration: --strict --build-label from source --build-date  --tcp-low-port 9000
--sge-parameter -pe smp $cores --strict --with-cvmfs-path /opt/libcvmfs --with-uuid-path /opt/uuid
--prefix /var/condor/execute/dir_2578/cctools-fb72a868-x86_64-centos7
```

How do workers find the executor?



Task execution model



Beware!

Tasks use entire worker on incomplete declarations

```
Worker has
available:

8 cores
512 MB of memory
512 MB of disk
```

```
Task a:

4 cores
100 MB of memory
```

```
Task b:

3 cores
100 MB of memory
```

Tasks a and b may NOT run in worker at the same time.
(disk resource is not specified.)

Create a worker (batch submission)

```
# using \ to break the command in multiple lines
# you can omit the \ and put everything in one line

# run 3 workers in condor, each of size 1 cores, 2048 MB
# of memory and 4096 MB of disk,
# to serve my-app
# and which timeout after 60s of being idle.

$ condor_submit_worker  --cores 1              \
                        --memory 2048          \
                        --disk   4096          \
                        -M my-app              \
                        --timeout 60           \
                        3
```

Work Queue Factory -- conf file

the configuration file can be modified while the factory is running

```
$ work_queue_factory -Tcondor -C my-conf.json
$ cat my-conf.json
{
    "master-name": "my-app",
    "max-workers": 200,
    "min-workers": 5,
    "workers-per-cycle": 5,
    "cores":  1,
    "disk":   10000,
    "memory": 4096,
    "timeout":   900,
    "tasks-per-worker": 4
}
```

What Work Queue does behind the scenes

1. Some tasks are run using full workers.
2. Statistics are collected.
3. Allocations computed to maximize throughput
    a. Run task using guessed size.
    b. If task exhausts guessed size, keep retrying on full (bigger) workers.
4. When statistics become out-of-date, go to 1.