ANNA WOODARD*, YADU BABUJI, KYLE CHARD, IAN FOSTER, DANIEL S. KATZ, MIKE WILDE, JUSTIN WOZNIAK

INTERACTIVE, SCALABLE, REPRODUCIBLE DATA ANALYSIS WITH CONTAINERS, JUPYTER, AND PARSL

THE UNIVERSITY OF CHICAGO
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Argonne NATIONAL LABORATORY

*annawoodard@uchicago.edu

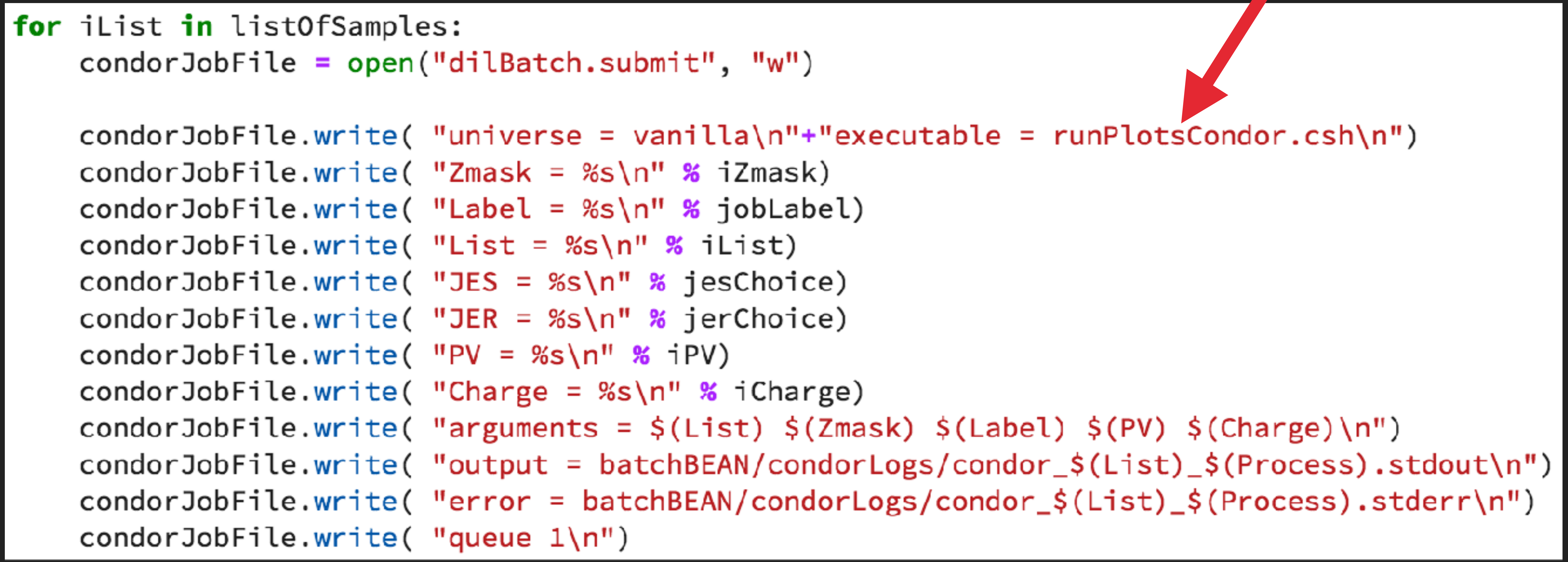
WHAT
IS
PARSL?

Parsl is a Python-based workflow system.

“TRADITIONAL” ANALYSIS BATCH SUBMISSION IN HEP

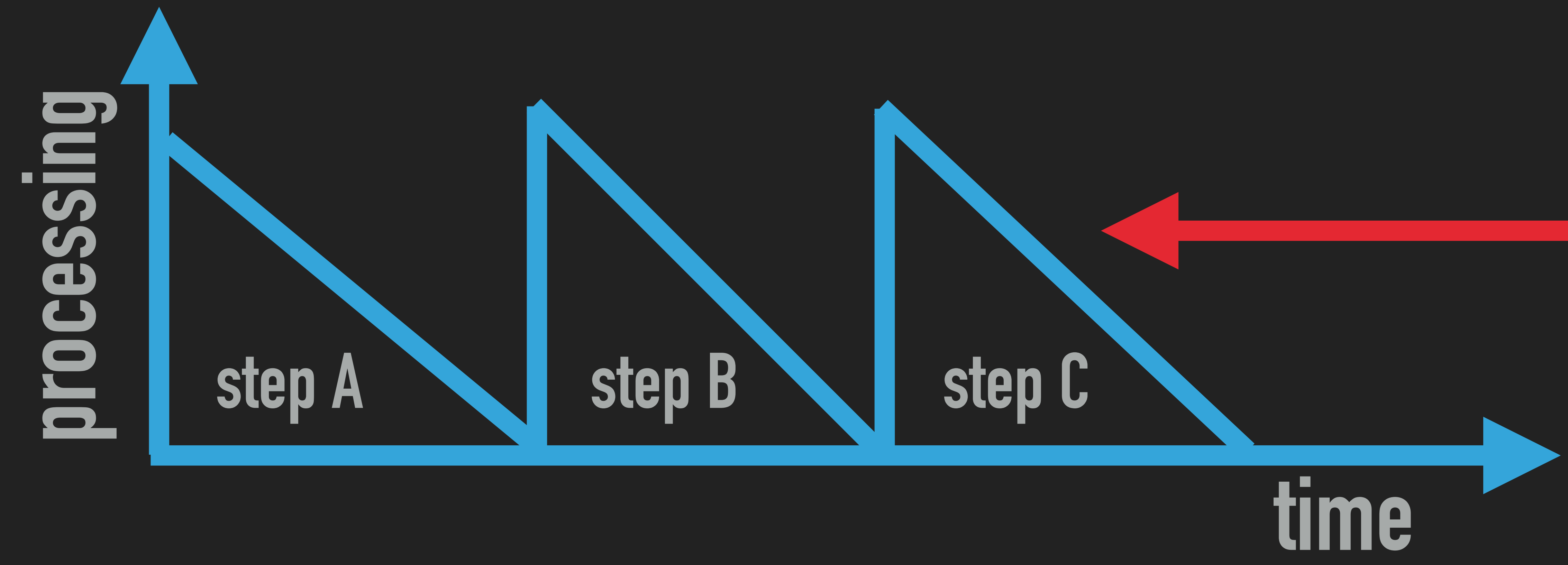
assumes fixed execution resources

manually produce submit script for specific Tier 2/ Tier 3



```
for iList in listOfSamples:
    condorJobFile = open("dilBatch.submit", "w")

    condorJobFile.write( "universe = vanilla\n"+"executable = runPlotsCondor.csh\n")
    condorJobFile.write( "Zmask = %s\n" % iZmask)
    condorJobFile.write( "Label = %s\n" % jobLabel)
    condorJobFile.write( "List = %s\n" % iList)
    condorJobFile.write( "JES = %s\n" % jesChoice)
    condorJobFile.write( "JER = %s\n" % jerChoice)
    condorJobFile.write( "PV = %s\n" % iPV)
    condorJobFile.write( "Charge = %s\n" % iCharge)
    condorJobFile.write( "arguments = $(List) $(Zmask) $(Label) $(PV) $(Charge)\n")
    condorJobFile.write( "output = batchBEAN/condorLogs/condor_$(List)_$(Process).stdout\n")
    condorJobFile.write( "error = batchBEAN/condorLogs/condor_$(List)_$(Process).stderr\n")
    condorJobFile.write( "queue 1\n")
```



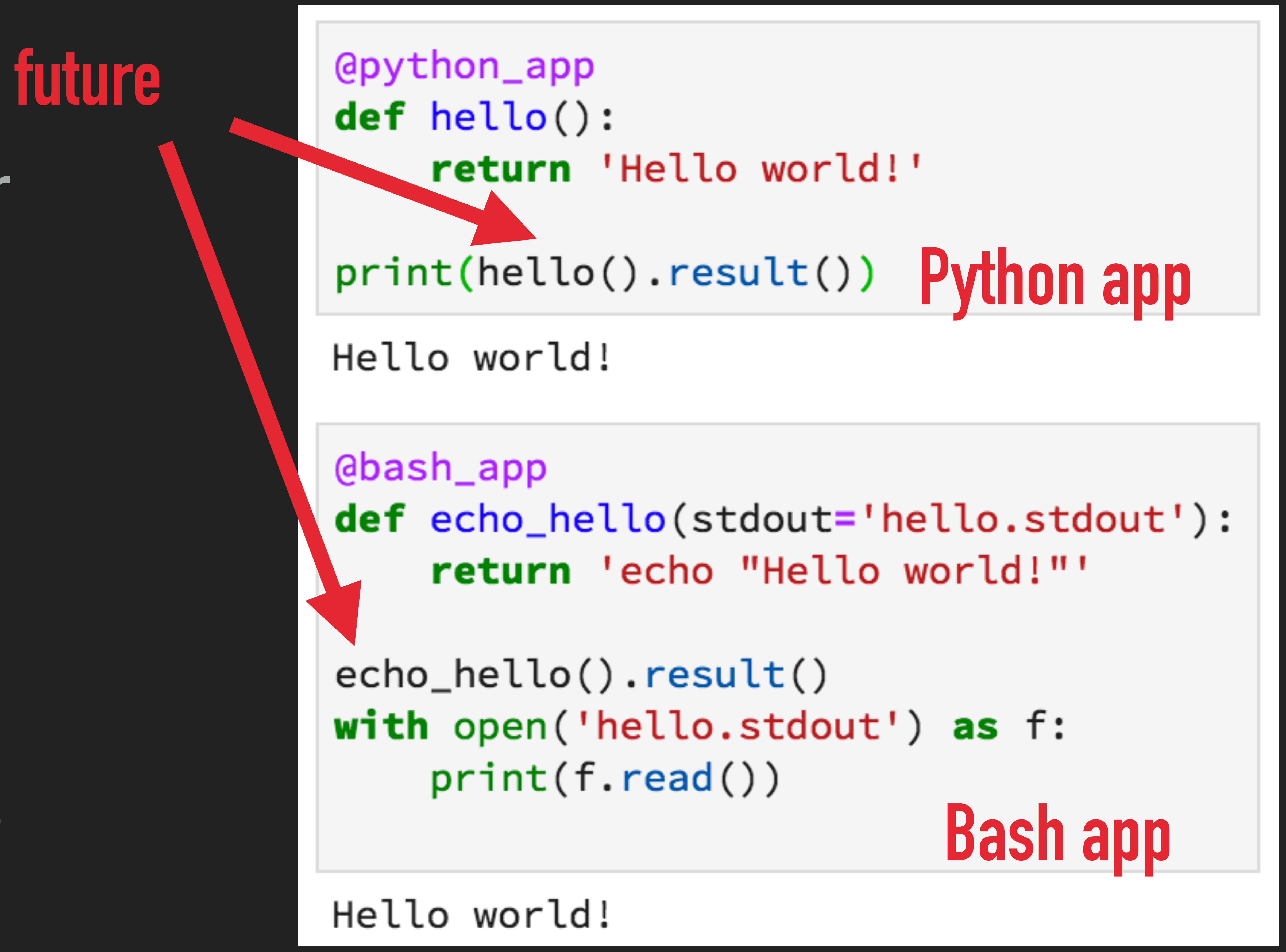
dependencies resolved by manually running steps sequentially

HOW IS PARSL DIFFERENT?

PARSL BASICS

pip install parsl

- Pure python; easy installation
- Rather than define code/input/output mapping externally, the user annotates functions to make Parsl apps
  - Bash apps call external applications
  - Python apps call Python functions
- Apps return “futures”: a proxy for a result that may not yet be available



future

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

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

Python app

```
Hello world!
```

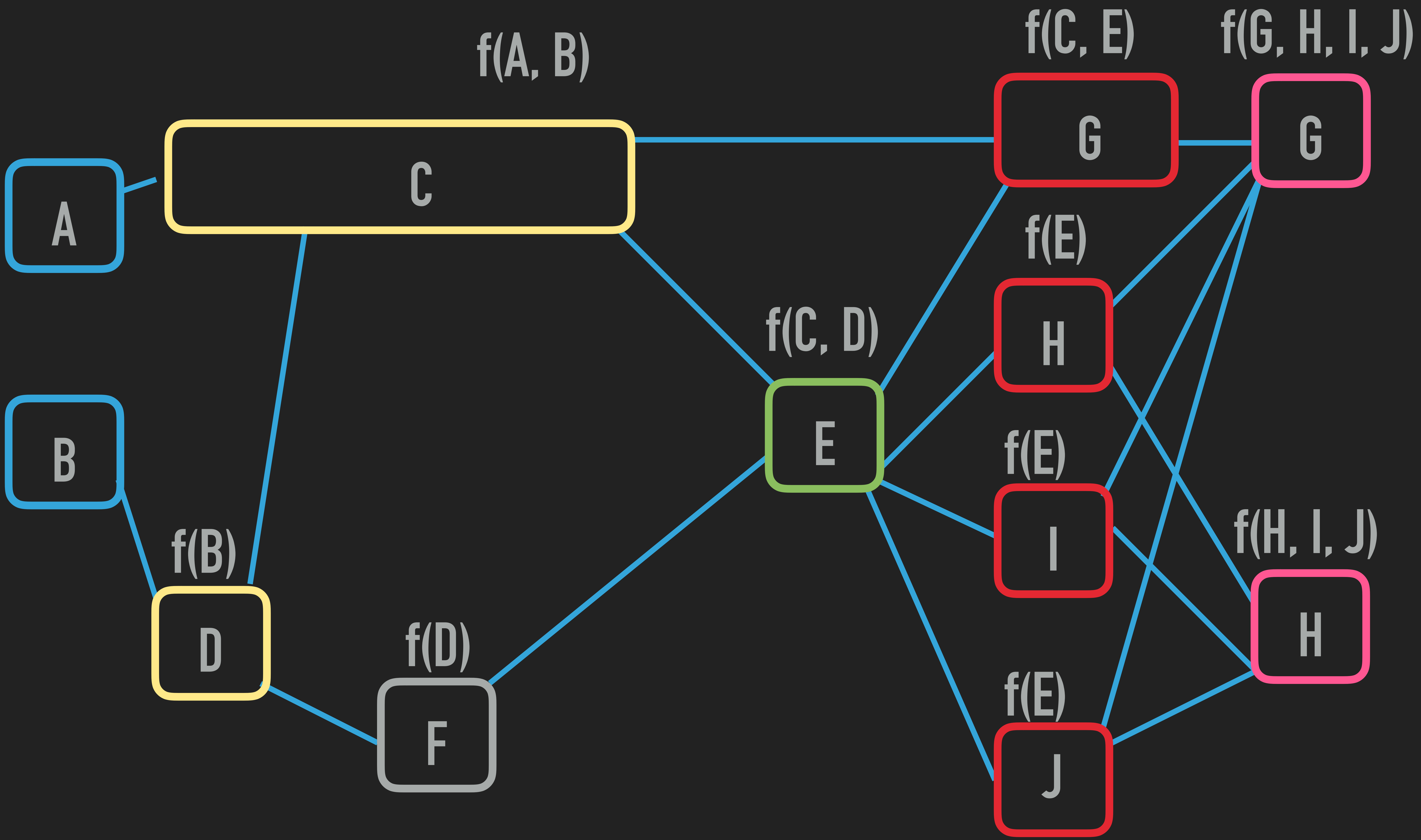
```
@bash_app
def echo_hello(stdout='hello.stdout'):
    return 'echo "Hello world!"'

echo_hello().result()
with open('hello.stdout') as f:
    print(f.read())
```

Bash app

```
Hello world!
```

RICH EXPRESSION OF DEPENDENCIES

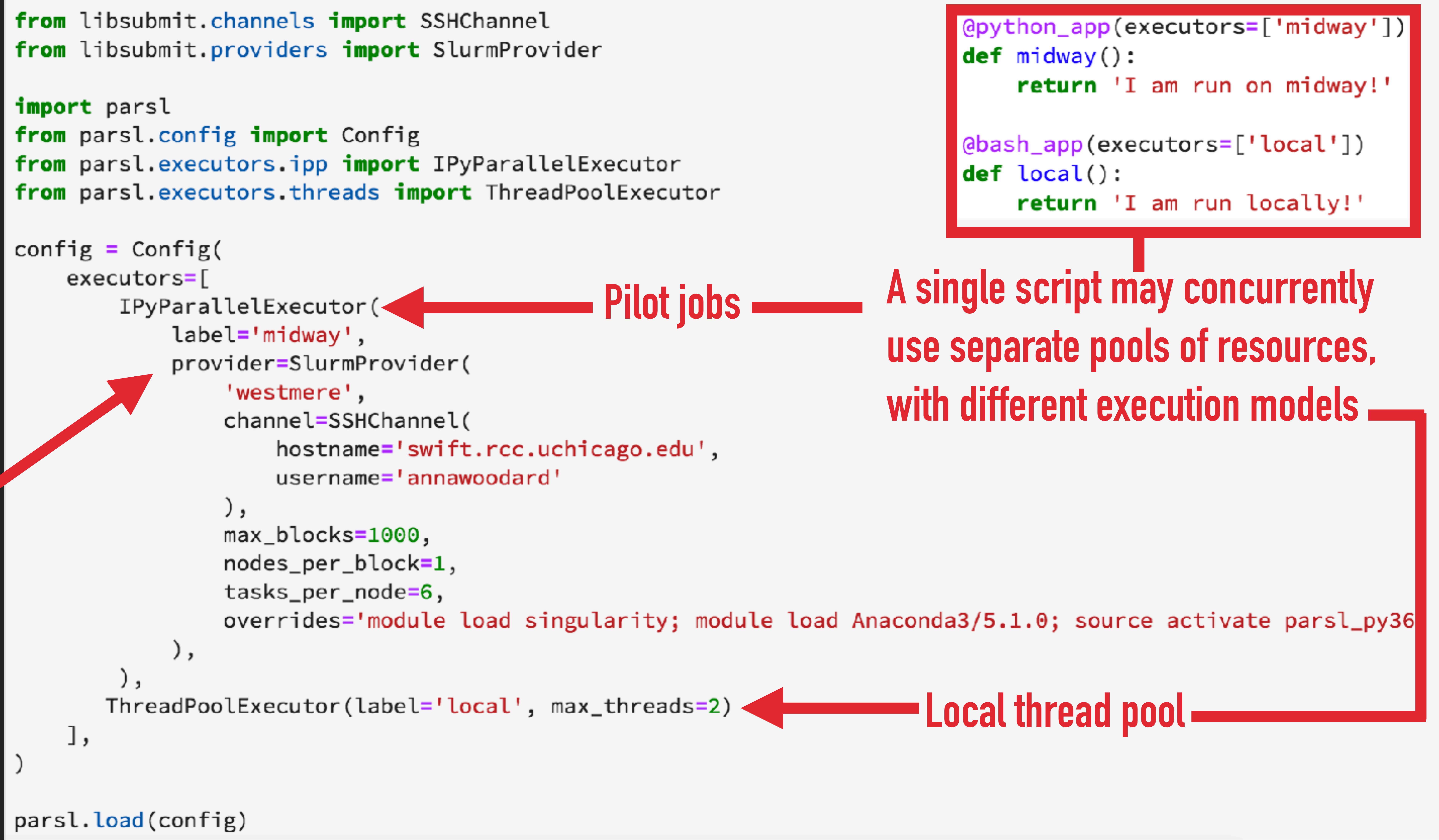


Apps run concurrently, respecting data dependencies via futures. Implicit parallel programming!

Dynamic: apps can create apps! Apps can be recursive!

SEPARATION OF CODE AND EXECUTION

Parsl scripts are independent of where they run. Write once, run the same script locally, on grids, clouds, or supercomputers!



```
from libsubmit.channels import SSHChannel
from libsubmit.providers import SlurmProvider

import parsl
from parsl.config import Config
from parsl.executors.ipp import IPyParallelExecutor
from parsl.executors.threads import ThreadPoolExecutor

config = Config(
    executors=[
        IPyParallelExecutor(
            label='midway',
            provider=SlurmProvider(
                'westmere',
                channel=SSHChannel(
                    hostname='swift.rcc.uchicago.edu',
                    username='annawoodard'
                ),
                max_blocks=1000,
                nodes_per_block=1,
                tasks_per_node=6,
                overrides='module load singularity; module load Anaconda3/5.1.0; source activate parsl_py36
            ),
        ),
        ThreadPoolExecutor(label='local', max_threads=2)
    ],
)

parsl.load(config)
```

```
@python_app(executors=['midway'])
def midway():
    return 'I am run on midway!'

@bash_app(executors=['local'])
def local():
    return 'I am run locally!'
```

Pilot jobs

A single script may concurrently use separate pools of resources, with different execution models

Local thread pool

Supported providers: AWS, Azure, Google Cloud, Slurm, Torque, HTCondor, Cobalt

*Note the format of this configuration will be supported in Parsl 0.6, being released this week.

PARSL FEATURES

- Apps can be shared as libraries
- Elasticity: resources used are scaled up and down according to demand automatically
- App caching and checkpointing: re-use results if app is called with the same inputs (record of inputs and outputs = provenance capture!)
- Workers can be launched in Docker containers (re-used for multiple apps); Docker/Shifter/Singularity/etc containers can be used with wrappers for per-app containerization
- Data transfer: Globus, HTTP, FTP

```
file = File(globus://endpoint/path/file)
```

PARSL IS ALREADY BEING USED IN A VARIETY OF DOMAINS.

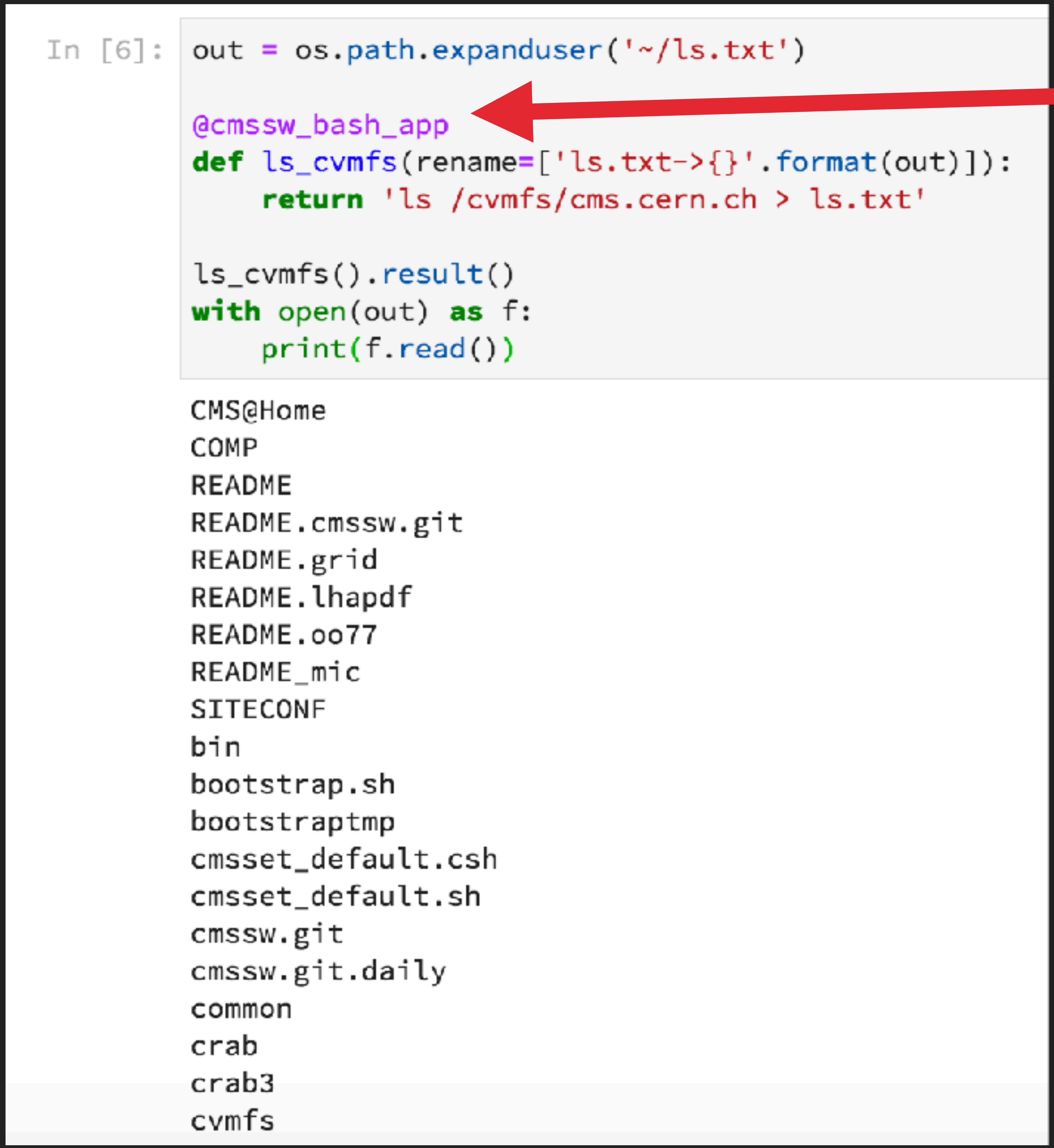
WHY NOT ADD HEP?

WHAT DO HEP TASKS NEED?

requirement	solution used
specific OS / run environment	vc3-builder (starts Singularity containers if they are needed) [1]
CVMFS mounted in userspace	Parrot (via vc3-builder) [2]
HEP software stack + user code	sandbox wrapper [3]

[1] http://virtualclusters.org
https://indico.cern.ch/event/587955/contributions/2937282/ (CHEP presentation, Kenyi Hurtado)
https://github.com/vc3-project/vc3-builder/
[2] https://ccl.cse.nd.edu/software/parrot/
[3] https://github.com/NDCMS/lobster

EXAMPLE IMPLEMENTATION FOR CMSSW+PARSL



In [6]:

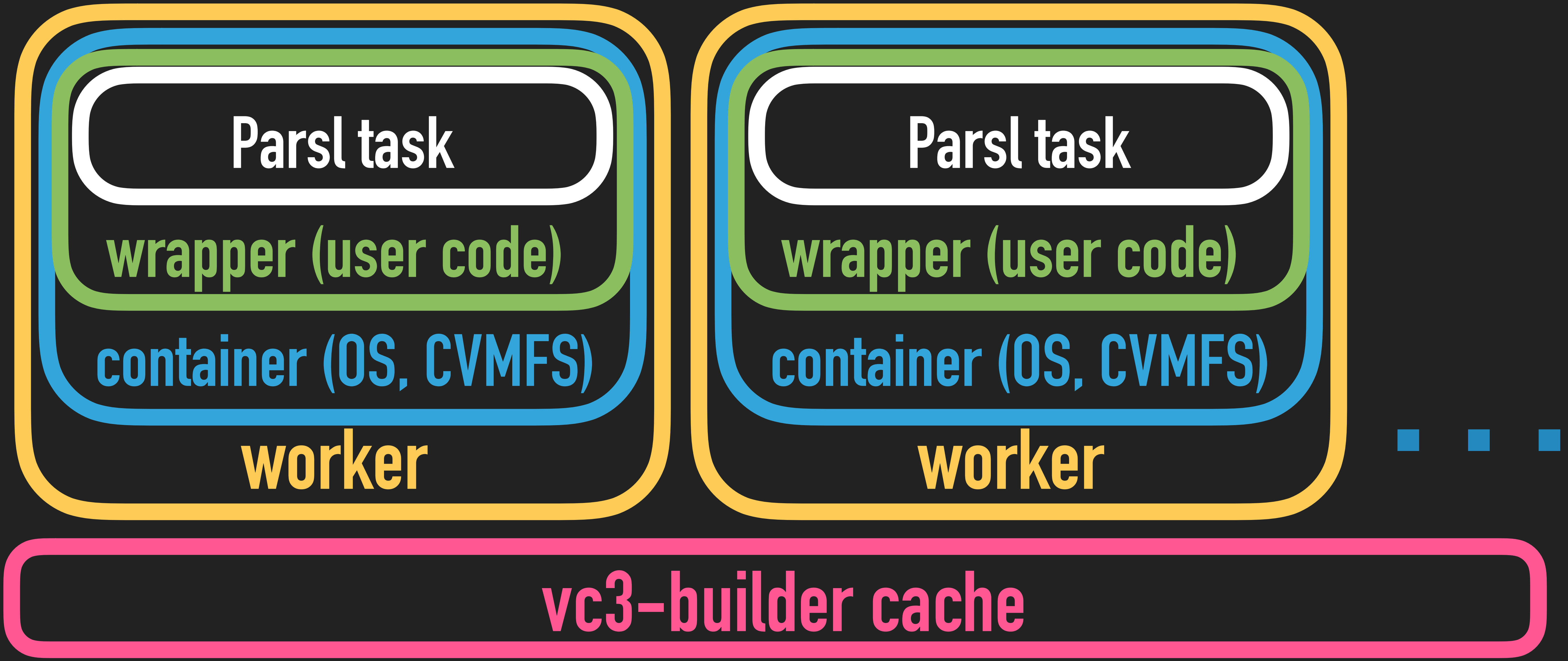
```
out = os.path.expanduser('~/ls.txt')

@cmssw_bash_app
def ls_cvmfs(rename=['ls.txt->{}'.format(out)]):
    return 'ls /cvmfs/cms.cern.ch > ls.txt'

ls_cvmfs().result()
with open(out) as f:
    print(f.read())
```

```
CMS@Home
COMP
README
README.cmssw.git
README.grid
README.lhapdf
README.oo77
README_mic
SITECONF
bin
bootstrap.sh
bootstraptmp
cmsset_default.csh
cmsset_default.sh
cmssw.git
cmssw.git.daily
common
crab
crab3
cvmfs
```

CMS-specific decorator (this is an example [1]; could be modified for other experiments)



Parsl task
wrapper (user code)
container (OS, CVMFS)
worker

Parsl task
wrapper (user code)
container (OS, CVMFS)
worker

. . .

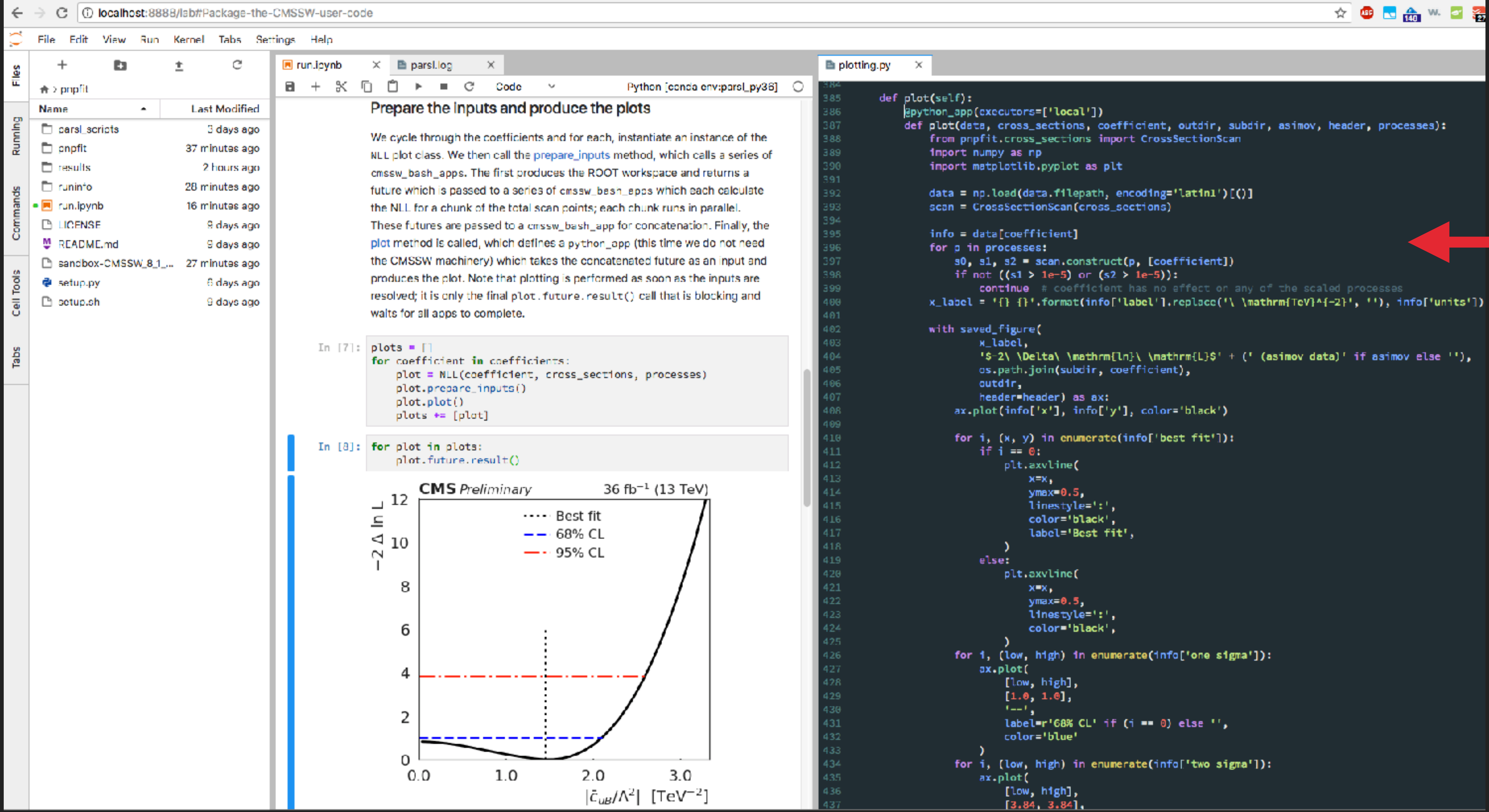
vc3-builder cache

[1] https://github.com/annawoodard/parslcms

WHY BOTHER WITH NOTEBOOKS?

- Traditional HEP analysis paradigm: code, results, and documentation are separate. Hard keep synchronized!
  - Notebooks allow these to be combined into a single narrative
- Web interface facilitates sharing
- Interactive plotting
- Native caching: fast, iterative development
- Jupyter Lab: text editors, terminals, data file viewers, and other custom components side by side with notebooks in a tabbed work area!
- Barriers to notebook adoption in HEP: complex software stacks, use of distributed computing

PARSL + NOTEBOOKS FOR HEP



Name	Last Modified
parsl_scripts	3 days ago
pnpfit	37 minutes ago
results	2 hours ago
runinfo	28 minutes ago
run.ipynb	16 minutes ago
LICENSE	9 days ago
README.md	9 days ago
sandbox-CMSSW_8_1_...	27 minutes ago
setup.py	6 days ago
setup.sh	9 days ago

Prepare the Inputs and produce the plots

We cycle through the coefficients and for each, instantiate an instance of the `NLL` plot class. We then call the prepare_inputs method, which calls a series of `cmssw_bash_apps`. The first produces the ROOT workspace and returns a future which is passed to a series of `cmssw_bash_apps` which each calculate the NLL for a chunk of the total scan points; each chunk runs in parallel. These futures are passed to a `cmssw_bash_app` for concatenation. Finally, the plot method is called, which defines a `python_app` (this time we do not need the CMSSW machinery) which takes the concatenated future as an input and produces the plot. Note that plotting is performed as soon as the inputs are resolved; it is only the final `plot.future.result()` call that is blocking and waits for all apps to complete.

In [7]:

```
plots = []
for coefficient in coefficients:
    plot = NLL(coefficient, cross_sections, processes)
    plot.prepare_inputs()
    plot.plot()
    plots += [plot]
```

In [8]:

```
for plot in plots:
    plot.future.result()
```

plotting.py

```
    def plot(self):
        @python_app(executors=['local'])
        def plot(data, cross_sections, coefficient, outdir, subdir, asimov, header, processes):
            from pnpfit.cross_sections import CrossSectionScan
            import numpy as np
            import matplotlib.pyplot as plt

            data = np.load(data.filepath, encoding='latin1')[()]
            scan = CrossSectionScan(cross_sections)

            info = data[coefficient]
            for p in processes:
                s0, s1, s2 = scan.construct(p, [coefficient])
                if not ((s1 > 1e-5) or (s2 > 1e-5)):
                    continue  # coefficient has no effect on any of the scaled processes
            x_label = '{} {}'.format(info['label'].replace('\ \mathrm{TeV}^{-2}', ''), info['units'])

            with saved_figure(
                    x_label,
                    '$-2\ \Delta\ \mathrm{ln}\ \mathrm{L}$' + (' (asimov data)' if asimov else ''),
                    os.path.join(subdir, coefficient),
                    outdir,
                    header=header) as ax:
                ax.plot(info['x'], info['y'], color='black')

                for i, (x, y) in enumerate(info['best fit']):
                    if i == 0:
                        plt.axvline(
                            x=x,
                            ymax=0.5,
                            linestyle=':',
                            color='black',
                            label='Best fit',
                        )
                    else:
                        plt.axvline(
                            x=x,
                            ymax=0.5,
                            linestyle=':',
                            color='black',
                        )
                for i, (low, high) in enumerate(info['one sigma']):
                    ax.plot(
                        [low, high],
                        [1.0, 1.0],
                        '--',
                        label=r'68% CL' if (i == 0) else '',
                        color='blue'
                    )
                for i, (low, high) in enumerate(info['two sigma']):
                    ax.plot(
                        [low, high],
                        [3.84, 3.84],
```

Real-world example [1] implementing part of my dissertation workflow in Parsl

[1] https://github.com/annawoodard/pnpfit/blob/master/run.ipynb

SUMMARY

- Parsl's implicit dataflow model allows for simple expression of complex dependencies
- In Parsl, code is separate from the specification of computing resources: this makes Parsl scripts portable and scalable
- Parsl has a number of useful features: app caching, elasticity, container support, data transfer, and more
- To extend Parsl for use in HEP, an example has been shown which wraps apps in a Singularity container with CVMFS mounted via VC3-builder, and the CMSSW user code set up via sandboxing. The workflow is orchestrated via a Jupyter notebook, which facilitates easy sharing and documentation, and fast iterative development.

STAY IN TOUCH!

Parsl

http://parsl-project.org

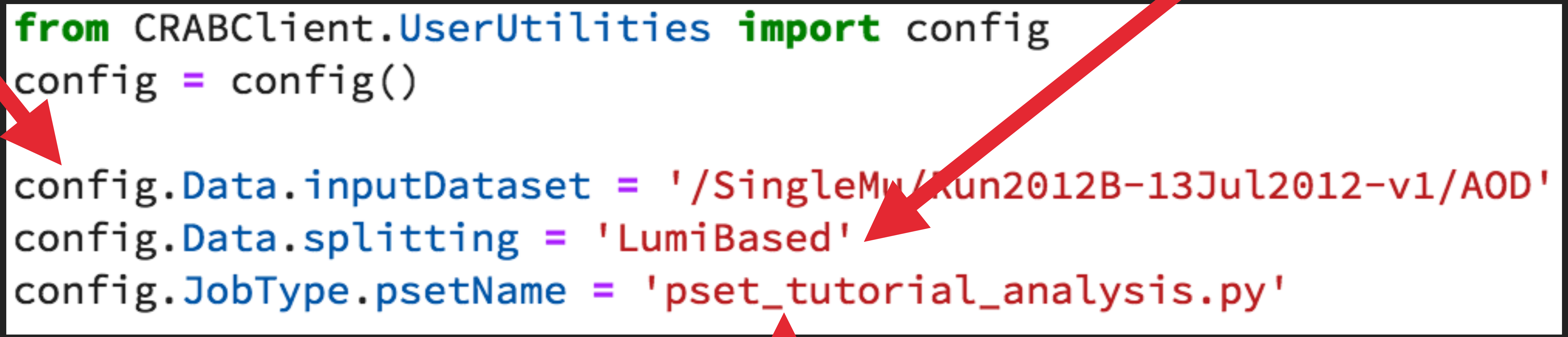
BACKUP

“TRADITIONAL” GRID SUBMISSION IN HEP

user explicitly defines:

input dataset

method for splitting dataset into chunks



```
from CRABClient.UserUtilities import config
config = config()

config.Data.inputDataset = '/SingleMu/Run2012B-13Jul2012-v1/AOD'
config.Data.splitting = 'LumiBased'
config.JobType.psetName = 'pset_tutorial_analysis.py'
```

code to execute on each chunk