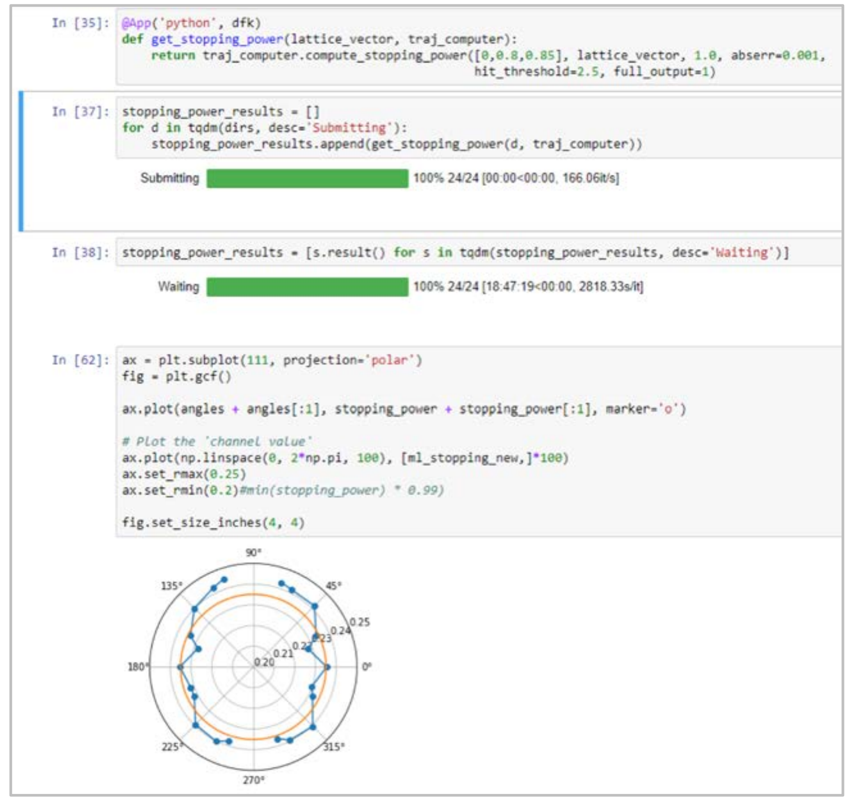
Understanding software sustainability: Learning from Parsl and other projects

Daniel S. Katz, dskatz@illinois.edu, d.katz@ieee.org, @danielskatz
Assistant Director for Scientific Software & Applications, NCSA
Research Associate Professor, CS, ECE, iSchool

ILLINOIS
NCSA | National Center for Supercomputing Applications

An increasingly common story…

- I'm developing an application and I need to link together external tools + functions
  - (where each tool is dependent on data from the previous tool)
- I have a notebook that does *X* and I need to run it on a cloud, cluster, supercomputer
- I need to run my analysis using a range of local and distributed datasets
- …
- And I want to do this in an interactive environment



Parsl: Interactive parallel scripting in Python

Annotate functions to make Parsl *apps*

- 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!

```
pip install parsl
```

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

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

```
Hello World!
```

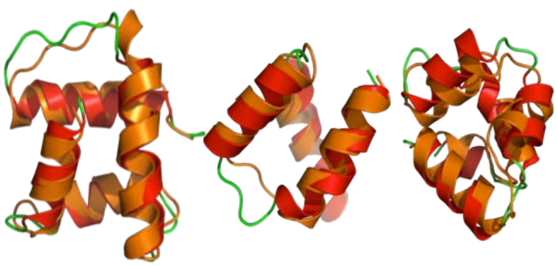
```python
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
     print(f.read())
```

```
Hello World!
```

When do you need automated workflow?

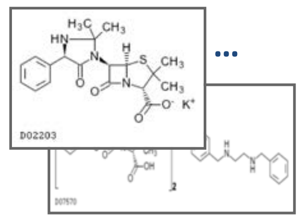
Example application: protein-ligand docking for drug screening

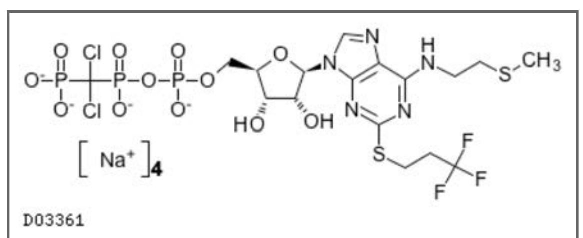
O(10) proteins implicated in a disease

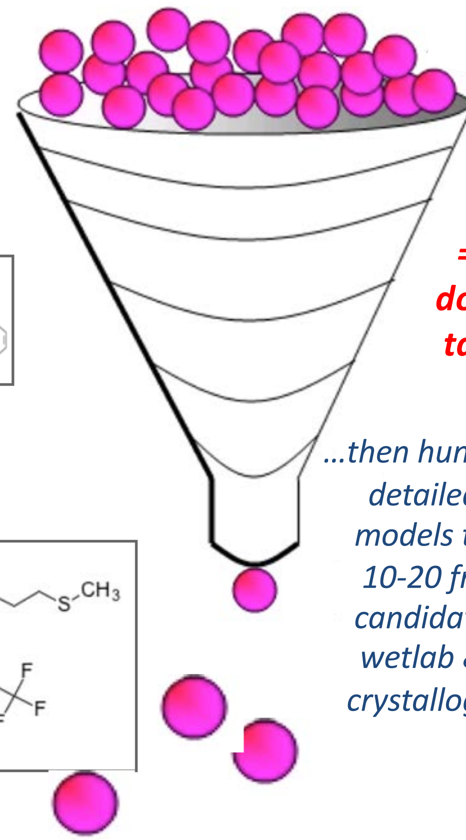


X

O(100K) drug candidates







= 1M docking tasks…

…then hundreds of detailed MD models to find 10-20 fruitful candidates for wetlab & APS crystallography

Expressing a many task workflow in Parsl

1) Wrap the protein docking code:

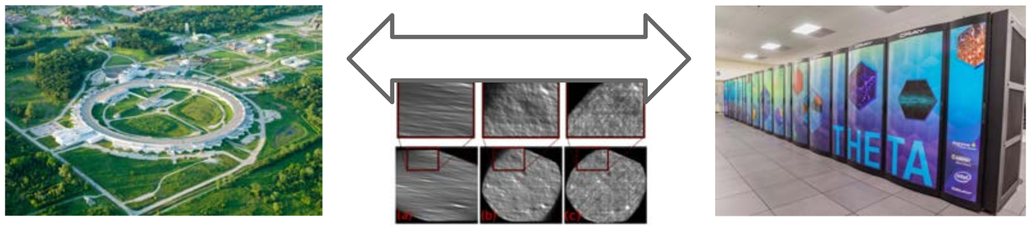
```
@bash_app
def dock(p, c, minRad, maxRad)
  return 'dock.sh {0} {1} {2} {3}'.format(p,
   c ,minRad, maxRad)
```

Expressing a many task workflow in Parsl

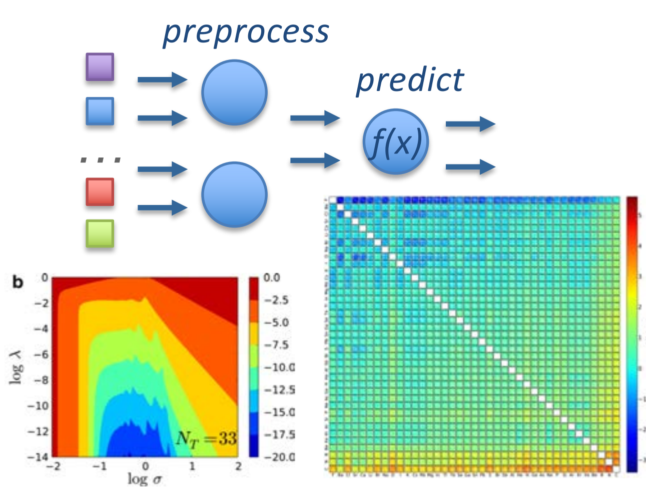
2) Execute the protein docking workflow:

```
for p in proteins:
   for c in ligands:
      structure[p][c] =
       dock(p, c, minRad, maxRad)

scatter_plot = analyze(structure)
```

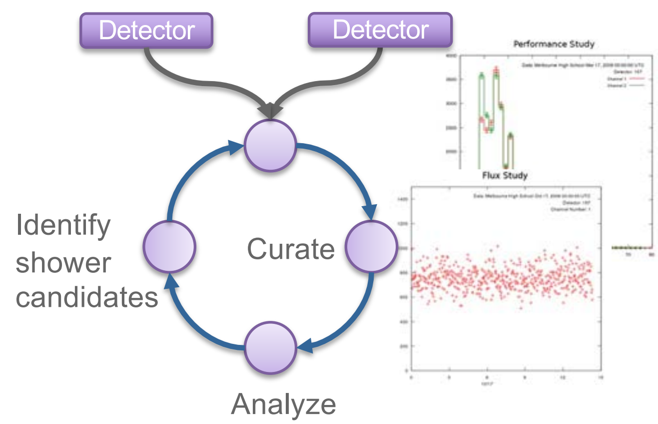
Workflows beyond batch computational science



Online computing



Machine learning



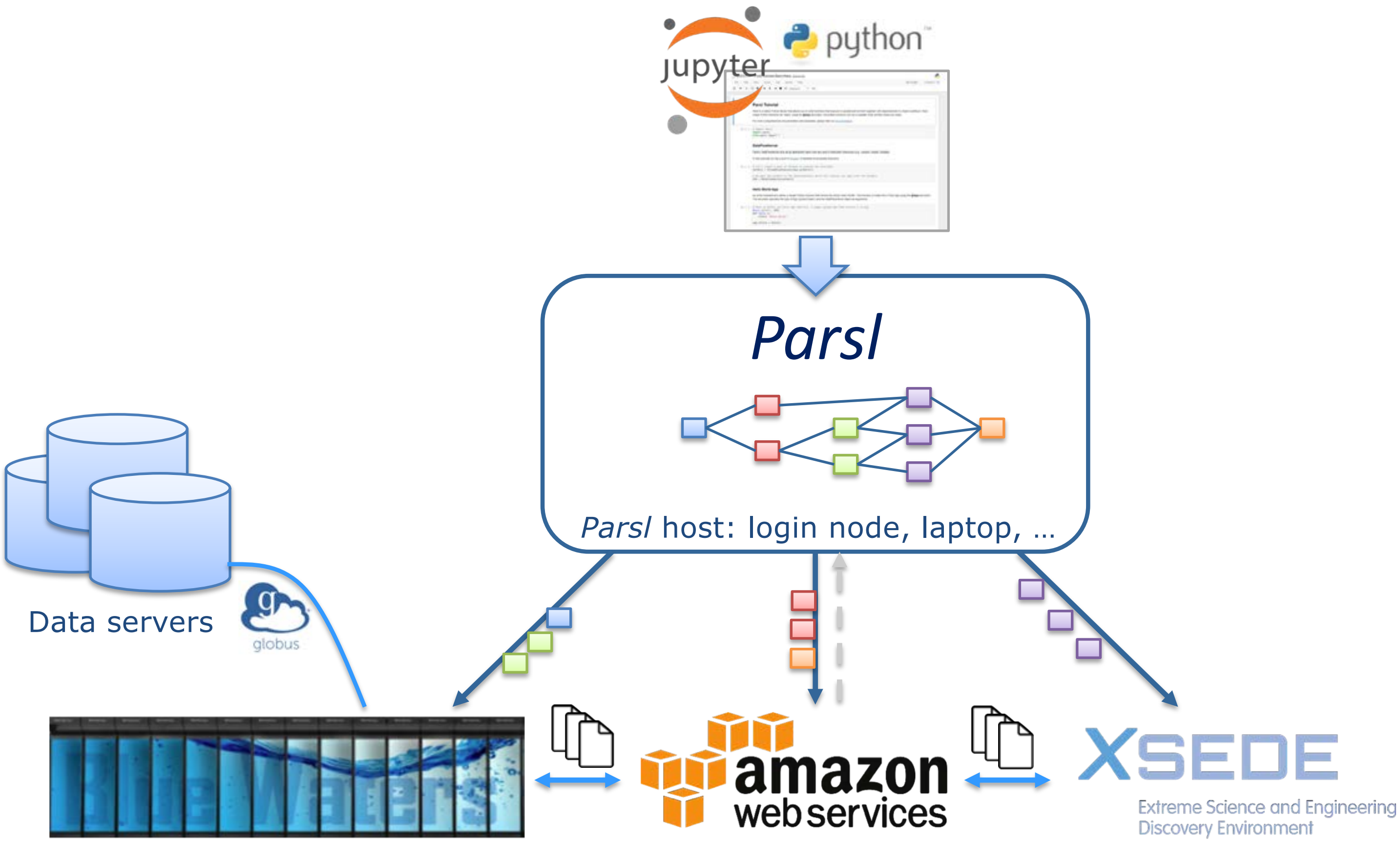
Interactive computing

Brief history: the Swift parallel scripting language

- 10+ years of development
- C-like language with implicit parallelism
- Applied in dozens of scientific domains
- Data management, multi-site execution, coasters, etc.
- Leveraging lessons and components to build Parsl

```
type file;

app (file o) simulation (int sim_steps, int sim_range, int sim_values)
{
  simulate "--timesteps" sim_steps "--range" sim_range "--nvalues" sim_values
stdout=filename(o);
}

app (file o) analyze (file s[])
{
  stats filenames(s) stdout=filename(o);
}

int nsim   = toInt(arg("nsim","10"));
int steps  = toInt(arg("steps","1"));
int range  = toInt(arg("range","100"));
int values = toInt(arg("values","5"));

file sims[];

foreach i in [0:nsim-1] {
  file simout <single_file_mapper; file=strcat("output/sim_",i,".out")>;
  simout = simulation(steps,range,values);
  sims[i] = simout;
}

file stats<"output/average.out">;
stats = analyze(sims);
```

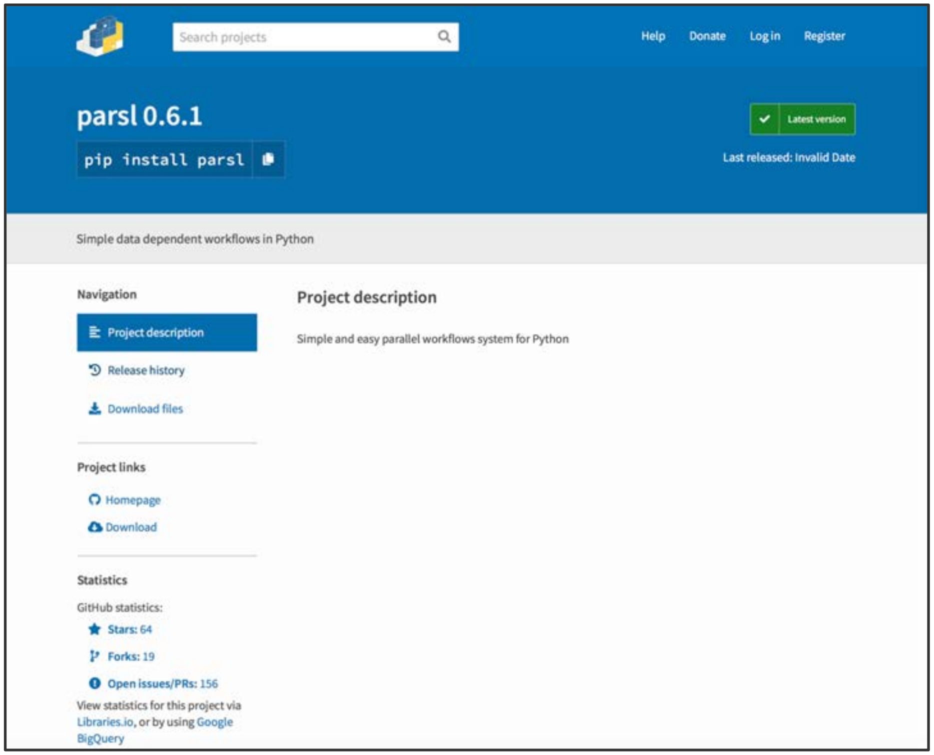
Parsl in action: dynamic dataflow execution



Parsl is Python

```
pip3 install parsl
```

- Use Python libraries natively
- Stage Python data transparently
- Integrates with Python ecosystem



Parsl scripts are execution provider independent

- The same script can be run locally, on grids, clouds, or supercomputers
  - Works directly with the scheduler (no HTC-like setup)
- Containers can be used for per-app execution or repeated invocation of the same app
- Currently supported execution providers:
  - Local, Cloud (AWS, private), Slurm, Torque, Condor, Cobalt

XSEDE

amazon web services

Open Science Grid

Google Cloud

Separation of code and execution

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

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

```python
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)
```

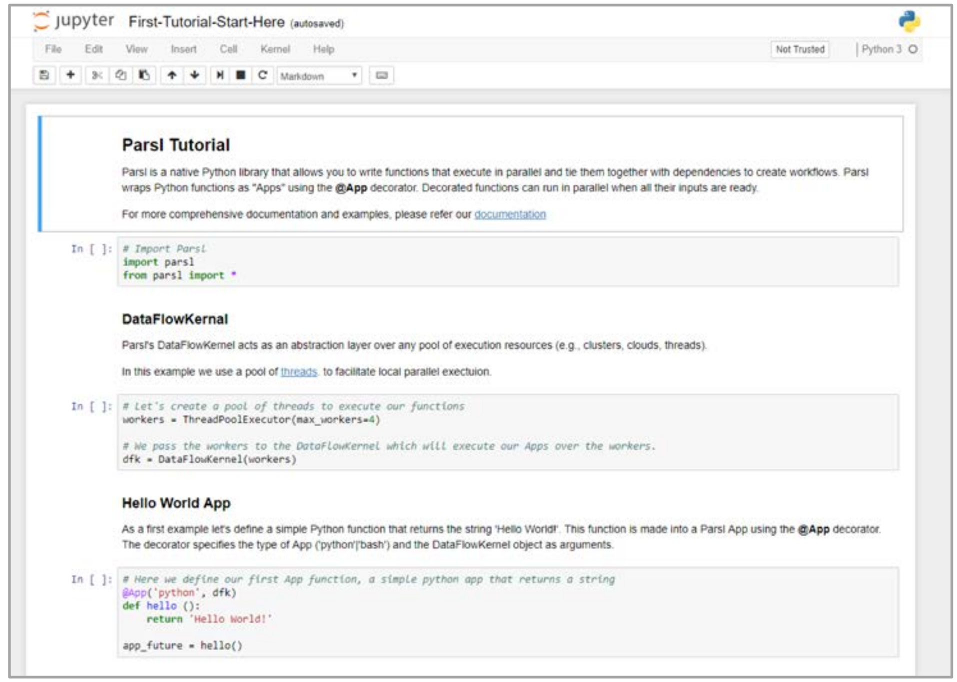
Pilot jobs on a cluster

Local threads

* Config format for Parsl 0.6

Interactive supercomputing in Jupyter notebooks

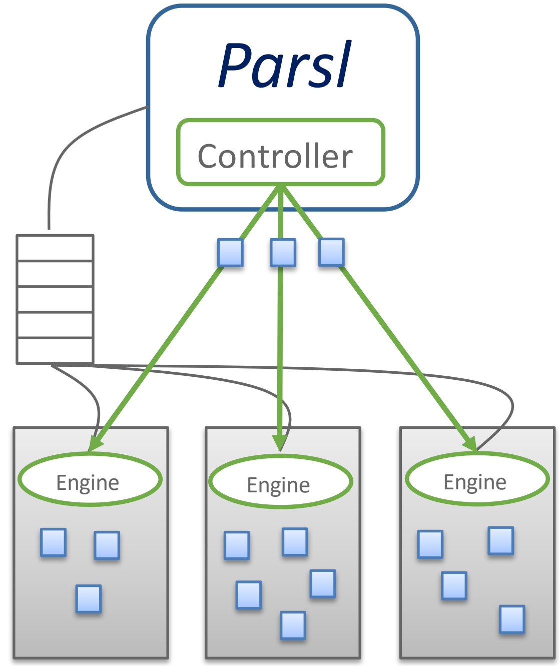
- Parsl can be used within a Jupyter notebook with no modifications necessary
- Tunneling and OAuth-based flows supports remote execution from the notebook
- Visualization of Parsl graph in notebook



A variety of execution models

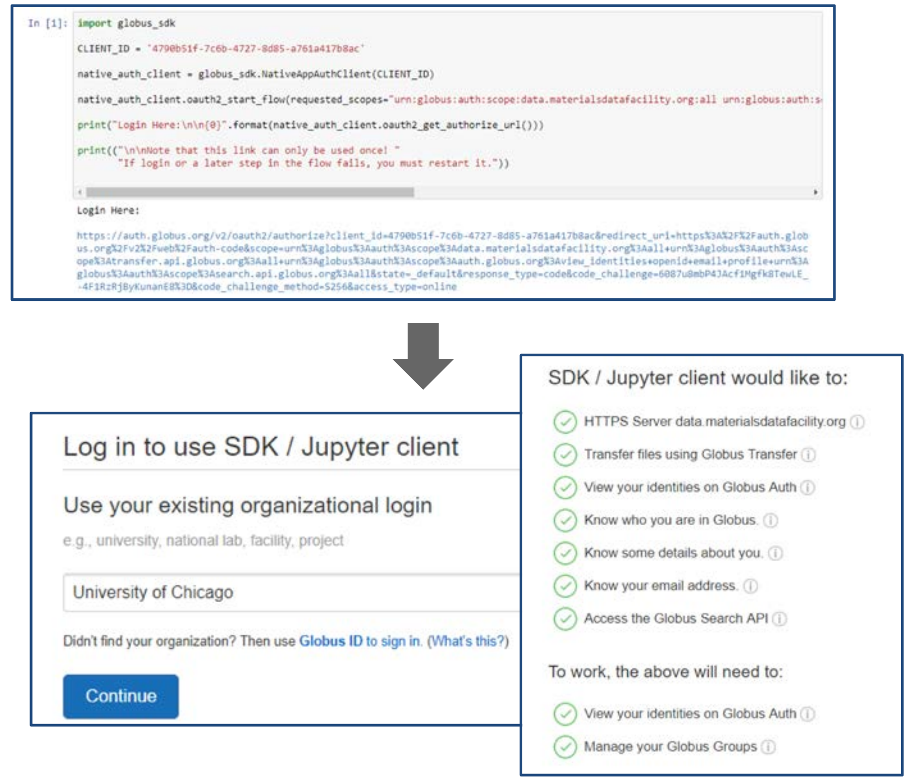
- Thread Pool
  - Local
- High throughput
  - Pilot job model
- Extreme scale
  - MPI-based pilot jobs

- New execution models can be added



Authentication and authorization

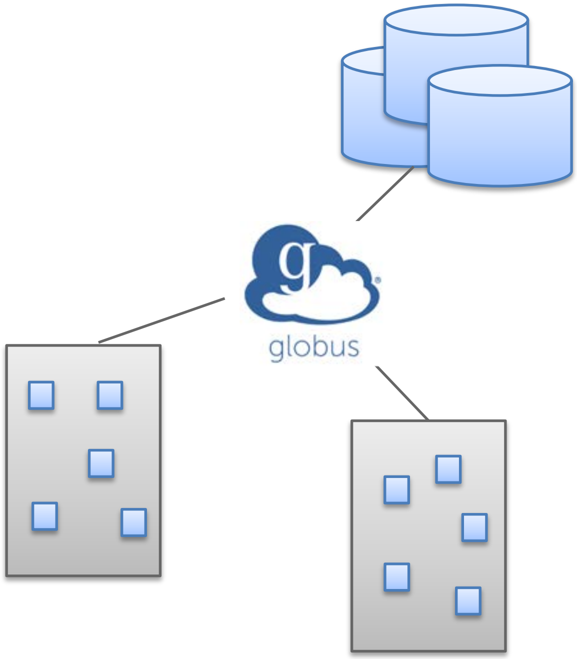
- A&A is hard today
  - 2FA, X509, etc.
- Integration with Globus Auth to support native app integration for accessing Globus (and other) services
- Using scoped access tokens, refresh tokens, delegation support



Transparent (wide area) data management

- Implicit data movement to/from repositories, laptops, supercomputers, …
- Globus for third-party, high performance and reliable data transfer
  - Support for site-specific DTNs
- HTTP/FTP direct data download/upload
- Compliments node-specific staging and caching models

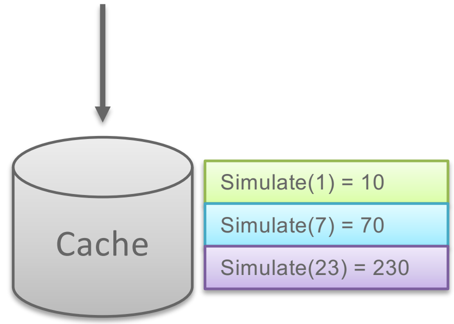
```
parsl_file =
   File(globus://EP/path/file)
```



App caching (memoization)

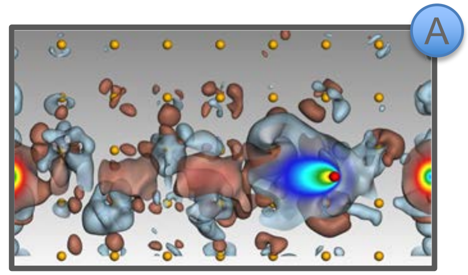
- Parsl apps are often expensive to recompute
- In many development modes results need not be recomputed
  - During development or interactive workflow
- Memoization optimizes execution by caching app results when called with the same inputs
- Parsl relies on user control to annotate deterministic functions

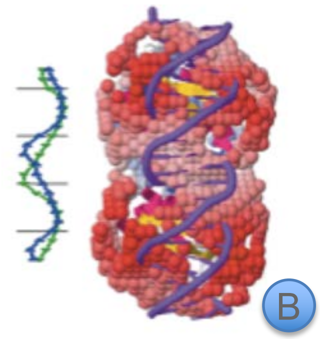
```
@python_app(cache=True)
def simulate(input_variable):
        return input_variable * 10
```

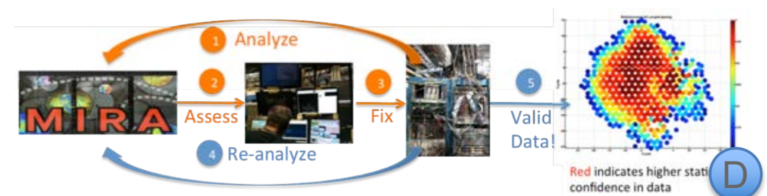


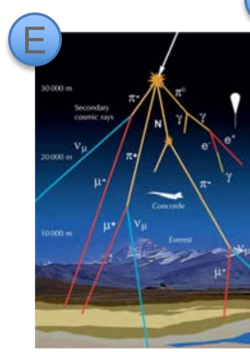
Scientific applications using Parsl

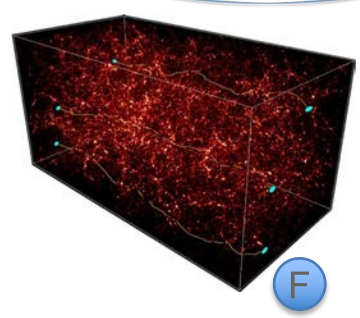
- (A) Machine learning to predict stopping power in materials
- (B) Protein and biomolecule structure and interaction
- (C) Information extraction to discovery facts in publications
- (D) Materials science at the Advanced Photon Source
- (E) Cosmic ray showers as part of QuarkNet
- (F) Weak lensing using sky surveys
- (G) Machine learning and data analytics (DLHub)

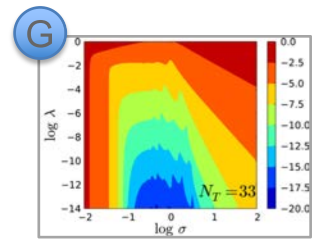












Parsl feature summary

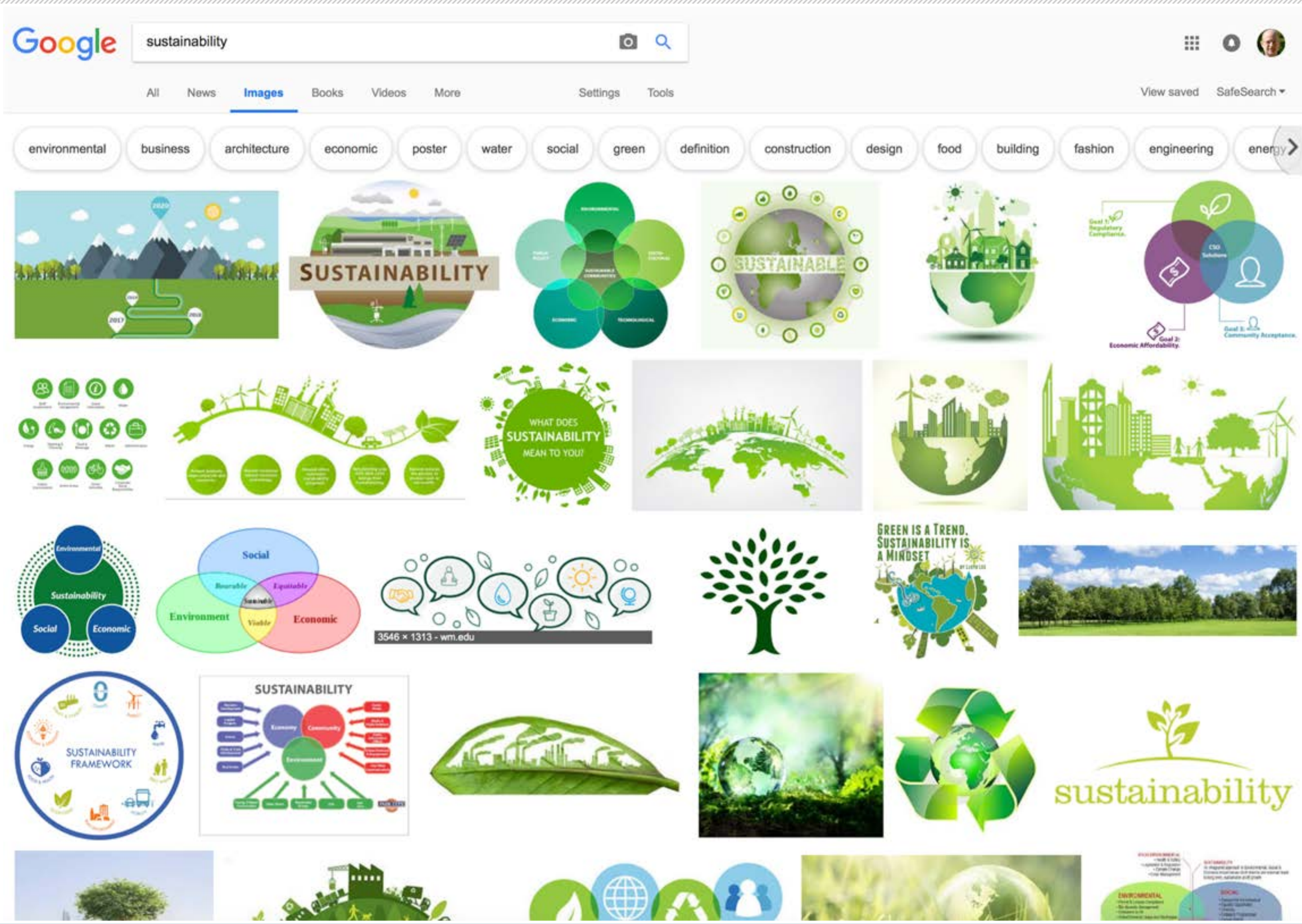
- Parsl's implicit dataflow model in Python allows for simple expression of complex dependencies
  - Expressed directly in Python
  - Can be used to implement a range of workflow models
- Parsl integrates with the scientific ecosystem
  - Development and execution of scalable applications in Jupyter
  - Use of common SciPy libraries
  - Integration with Globus
- In Parsl, code is separate from the specification of computing resources and data location: this makes Parsl scripts portable and scalable
- Parsl has a number of other important features:
  - app caching, checkpointing, elasticity, container support, data transfer, and more

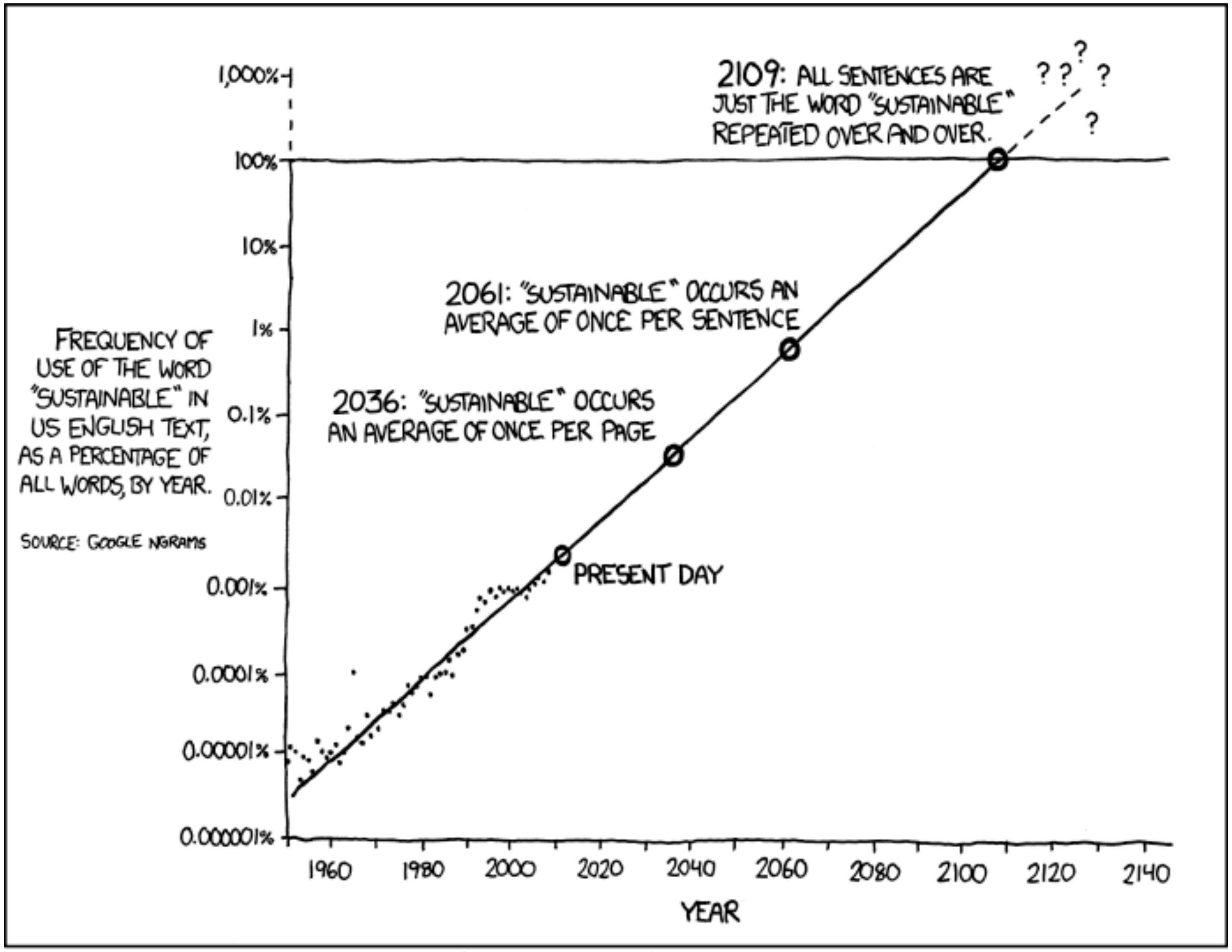
Parsl project summary

- Initially funded by NSF, $3m over 3 years (stretched to 4)
- 2.5 core developer FTEs, PI, co-PIs, chemistry & education application developers, undergraduate & graduate students
- Open source, intended as open community, including library of reusable workflows
- Some success with purely external contributions to code
- More success with collaborating projects

- What happens next? How we make Parsl sustainable?

What is sustainability?


Google
sustainability
All
News
Images
Books
Videos
More
Settings
Tools
View saved
SafeSearch
environmental
business
architecture
economic
poster
water
social
green
definition
construction
design
food
building
fashion
engineering
SUSTAINABILITY
SUSTAINABLE
WHAT DOES SUSTAINABILITY MEAN TO YOU?
Environmental
Sustainability
Social
Economic
Social
Bearable
Equitable
Environment
Viable
Economic
3546 × 1313 - wm.edu
GREEN IS A TREND. SUSTAINABILITY IS A MINDSET
SUSTAINABILITY FRAMEWORK
SUSTAINABILITY
sustainability



THE WORD "SUSTAINABLE" IS UNSUSTAINABLE.

What is sustainability?

- Most often used in the context of ecology, often specifically in the relationship between humans and the planet
- Example: Karl-Henrik Robèrt (via Wikipedia & paraphrased)
  - Natural processes are cyclical but we process resources linearly
  - We use up resources, resulting in waste
  - Waste doesn’t find its way back into natural cycles; not reused or reassimilated
  - Call for "life-styles and forms of societal organization based on cyclic processes compatible with the Earth's natural cycles"

Software sustainability

Software sustainability for whom?

- (Parsl) Users
- (Parsl) Funders
- (Parsl) Managers
- (Parsl) Developers (& Maintainers)

Software sustainability for users

- The capacity of the software to endure
- Will the software (Parsl) will continue to be available in the future, on new platforms, meeting new needs?

- Really:
  - Shopping
  - With elements of
    - Longevity
    - Robustness
    - Support

Software sustainability for funders

- My definition while an NSF program officer:
- “If I give you funds for this (Parsl) now, how will you keep it going after these funds run out?”
- “… without coming back to me for more funds”

- Really
  - Portfolio management

Software sustainability for managers

- Focused on people, not software
- How do I keep the (Parsl) team going?

- Really:
  - Business
  - Capitalism
  - Entrepreneurship

Software sustainability for developers

- Often focused on resources, not software
  - How do I get the resources needed to keep my (Parsl) software alive and up-to-date?
  - And keep myself supported / employed?
- Counterpart
  - How do I make keeping my software alive and up-to-date use less resources?

- Really
  - Entrepreneurship
  - Community building
  - Software engineering

Software collapse[1]

- Software stops working eventually if is not actively maintained
- Structure of computational science software stacks:
  1. Project-specific software (developed by researchers): software to do a computation using building blocks from the lower levels: scripts, workflows, computational notebooks, small special-purpose libraries & utilities
  2. Discipline-specific software (developed by developers & researchers): tools & libraries that implement disciplinary models & methods
  3. Scientific infrastructure (developed by developers): libraries & utilities used for research in many disciplines
  4. Non-scientific infrastructure (developed by developers): operating systems, compilers, and support code for I/O, user interfaces, etc.
- Software builds & depends on software in all layers below it; any change below may cause collapse

[1]http://blog.khinsen.net/posts/2017/01/13/sustainable-software-and-reproducible-research-dealing-with-software-collapse/

Software collapse[1]

- Options similar for house owners facing the risk of earthquakes:
  1. Accept that your house or software is short-lived; in case of collapse, start from scratch
  2. Whenever shaking foundations cause damage, do repair work before more serious collapse happens
  3. Make your house or software robust against perturbations from below
  4. Choose stable foundations
- Very short term projects might do 1 (code and throw away)
- Most active projects choose 2 (sustainability work)
- We don’t know how to do 3 (CS research needed, maybe new thinking)
- 4 is expensive & limits innovation in top layers (banks, military, NASA)

[1]http://blog.khinsen.net/posts/2017/01/13/sustainable-software-and-reproducible-research-dealing-with-software-collapse/

Common elements

- Due to software collapse, bugs, new use cases, there are lots of risks to all parties
  - Users want to make good product choices that pay off in discoveries
  - Funders want to make good investments that pay off in discoveries
  - Managers want to keep staff employed, also create discoveries
  - Developers want their software to be used in discoveries (and want a career)
- (Almost) all want to know, will this software work in the future?
  - What’s the risk?
  - And how do developers get recognized?

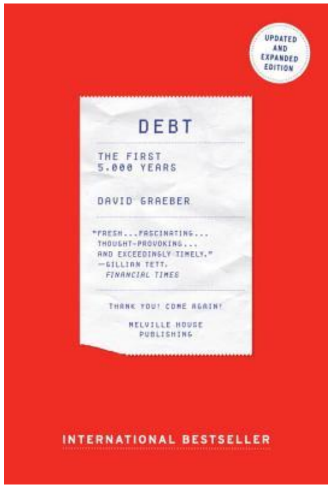
Back to sustainability, in the context of software

- Elinor Ostrom's (Governing the Commons) definition of sustainability for a common-pool resource (CPR): "As long as the average rate of withdrawal does not exceed the average rate of replenishment, a renewable resource is sustained over time."
  - Notion of a cyclic property, though cycle period not specified
  - But rate of what?
- Titus Brown[1]: "the common pool resource in open online projects is effort"
- Sustainability of effort may be appropriate for the developer
  - For effort to be available, need link to recognition, reward, position
- Sustainability of software may be appropriate for the user and funder
  - Rate of what?
- Sustainability of funding may be appropriate for the manager
  - Also helps developers
  - Rate of funding?

[1]A framework for thinking about Open Source Sustainability? http://ivory.idyll.org/blog/2018-oss-framework-cpr.html

"Equations" of software sustainability

- Software sustainability ≡ sufficient Δ software state
  - Sufficient to deal with: software collapse, bugs, new features needed
- Δ software state = (human effort in – human effort out - friction) * efficiency
  - Software stops being sustained when human effort out > human effort in over some time
- Human effort $\leftrightarrows$ $
  - All human effort works (community open source)
  - All $ (salary) works (commercial software, grant funded projects)
  - Combined is hard, equation is not completely true, humans are not purely rational
- Δ software state $\overset{?}{\rightarrow}$ users choose to volunteer effort or $
  - Development choices might take this into account



Debt: The First 5,000 Years
by David Graeber

Software sustainability and time

- Software sustainability is a measure of a dynamic, (unpredictable), time domain system
  - Back to risk…
- Software sustainability is a prediction – it can’t be known with certainty
- Software sustainability can only be measured looking backward
  - How do we know that software is no longer sustainable / has stopped being sustained?
    - It no longer works at all? (continuous integration fails)
    - It’s not being actively maintained? (no commit in the last x months)
    - It’s not being actively developed? (no non-bug fix commit in the last x months)
- What do we do for similar measures in other fields?
  - Guess (aka estimate)
  - Based on past performance
  - E.g., Project cost
- Research is needed

Summary

- Parsl as an example of an open source project
  - Started with funding and a core team
  - Will need to expand to be a community project and consider how to bring in new resources (funds or people)
- Software sustainability means different things to different groups of people
  - Persistence of working software
  - Persistence of people (or funding)
- Can define sustainability as
  - Inflow of resources is sufficient to do the needed work
  - Those resources can be turned into human effort
- In all cases, sustainability is not possible to measure in advance
  - Can only measure looking backward
  - Looking forward, can only predict

Acknowledgements

- Parsl (http://parsl-project.org) team: Yadu Babuji, Kyle Chard, Ben Clifford, Ian Foster, Lukasz Lacinski, Zhuozhao Li, Connor Pigg, Michael Wilde, Anna Woodard, Justin Wozniak
- Y. Babuji, A. Brizius, K. Chard, I. Foster, D. S. Katz, M. Wilde, J. Wozniak, "Introducing Parsl: A Python Parallel Scripting Library," 2018. https://doi.org/10.5281/zenodo.853491
- Discussions with: Neil Chue Hong and the UK SSI; Rob Haines and Caroline Jay at U. Manchester
- Keynote by James Howison at RSE2018
- Feedback from Matt Turk, James Howison, Dan Sholler

- D. S. Katz, "Scientific Software Challenges and Community Responses," 2015. https://www.slideshare.net/danielskatz/scientific-software-challenges-and-community-responses
- C. C. Venters, C. Jay, L. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, J. Austin, C. E. Dibsdale, J. Xu, "Software Sustainability: The Modern Tower of Babel," Proceedings of Third International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy 2014), Karlskrona, Sweden. http://ceur-ws.org/Vol-1216/paper2.pdf
- C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, C. C. Venters, "Sustainability design and software: The Karlskrona manifesto," 37th International Conference on Software Engineering (ICSE'15), 2015. https://doi.org/10.1109/ICSE.2015.179
- P. Johnston, M. Everard, D. Santillo, and K.-H. Robèrt, "Reclaiming the Definition of Sustainability," *Environmental Science and Pollution Research*, v.14(1), pp. 60-66, 2007. https://doi.org/10.1065/espr2007.01.375

ILLINOIS

NCSA | National Center for
Supercomputing Applications