A Python Library for Automatic Geospatial Data Harmonization and Seamless Transition to Cloud-Based Processing

J. Bieszczad*, M. Shapiro*, D. Entekhabi†, D. Callender*, J. Milloy*, D. Sullivan*, M. Ueckermann*

*Creare LLC, † Massachusetts Institute of Technology

Motivation

- Earth data scientists **want to**:
  - **Find** data relevant to a scientific inquiry
  - **Explore** data to discover a new characteristic or feature
  - **Produce** new data products and analyses
  - **Validate** the results of a simulation or analysis
  - **Answer** scientific questions
  - **Share** results with the community
- However, data scientists **have to** deal with large scale data **variety** and **volume** challenges

Project Objectives

- Develop PODPAC, an open-source, Python library, which **removes major barriers to widespread exploitation of earth science data**
- Automate geo-data wrangling for **integrated analyses of disparate data sources** in a plug-and-play manner
- Enable data scientists to **easily transition workstation analyses to massively distributed processing** on Amazon Web Services (AWS)
- Facilitate **generation and sharing** of reproducible and documented earth science data products and algorithms

Open Source Development

- PODPAC is free and open-source software available at https://podpac.org/
- **Wanted**: Testers, early adopters, contributors and feedback

Acknowledgment

- This research is supported by NASA under SBIR Phase II Contract No. 80NSSC18C0061

Pipeline for Observational Data Processing, Analysis, and Collaboration

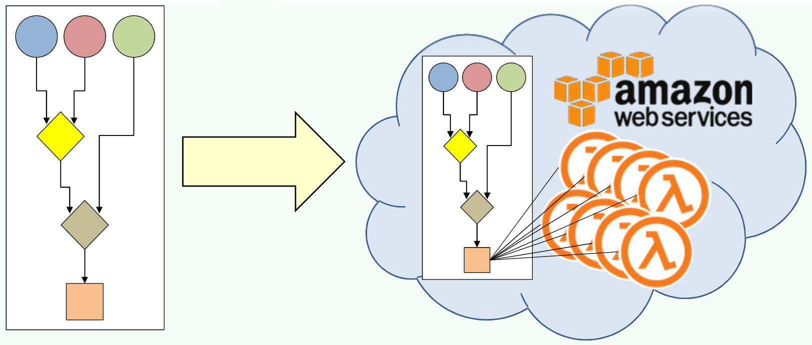
Readily develop integrated geospatial analyses and analytics on your workstation

Seamlessly transition to scalable, massively distributed processing on the cloud

Encapsulated Local and Remote Data Sources

Plug-and-Play User Algorithms

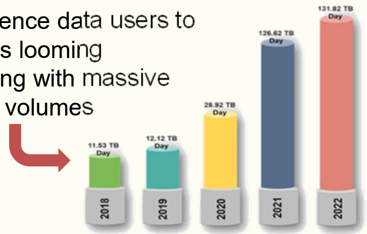
Documented Earth Science Data Products



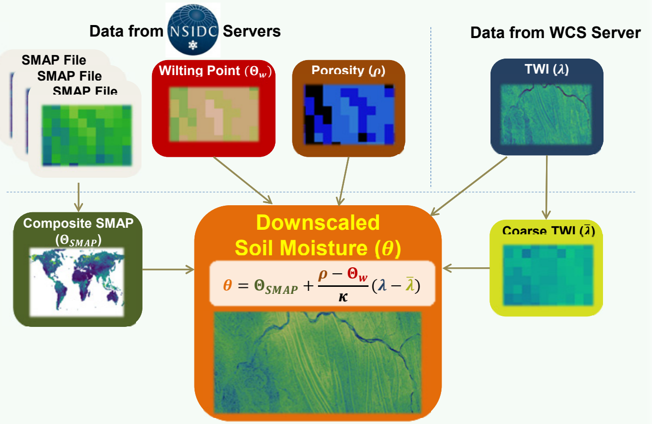
- ✓ Use of Python and Jupyter Notebooks reduces software learning curve for new users
- ✓ Local and remote data (OPeNDAP, WCS, etc.) are encapsulated in common API wrapper for plug-and-play integration within user-specified algorithms
- ✓ Automated data wrangling handles differences in geospatial CRS, projections, resolution, formats, etc.

- ✓ Generated data products automatically record data provenance (sources, algorithms, versions) for reproducibility and documentation via JSON metadata

- ✓ JSON metadata enables direct deployment and execution of PODPAC algorithm pipelines on AWS
- ✓ PODPAC-enabled "serverless" AWS Lambda functions avoid provisioning and maintenance of cloud servers
- ✓ PODPAC Lambda functions automatically scale up to 1024 parallel computational processes
- ✓ Processing on AWS "close to data storage" improves performance and avoids costly egress charges
- ✓ Migrating earth science data users to AWS helps address looming challenges in dealing with massive earth science data volumes



"Serverless" SMAP Downscaling



$$\theta = \theta_{SMAP} + \frac{\rho - \theta_w}{\kappa}(\lambda - \bar{\lambda})$$

SMAP Downscaling Using PODPAC in a Jupyter Notebook

Create PODPAC nodes for accessing NSIDC data via OPeNDAP

```
In [2]:
import podpac  # Make the "PODPAC" library available to Python
smap = podpac.datalib.smap.SMAP(interpolation='bilinear', cache_type='disk')
wilt = podpac.datalib.smap.SMAPWilt(interpolation='bilinear', cache_type='disk')
porosity = podpac.datalib.smap.SMAPPorosity(interpolation='bilinear', cache_type='disk')
```

Create PODPAC node to access topographic wetness index (TWI) from a WCS server

```
In [3]:
twi = podpac.data.WCS(source=podpac.utils.load_setting('WCS_URL'),
                      layer_name=podpac.utils.load_setting('TWI'),
                      interpolation='nearest')
```

Reproject high resolution TWI onto low resolution SMAP grid

```
In [4]:
twi_bar = podpac.data.ReprojectedSource(source=twi,
                                        reprojected_coordinates=smap.shared_coordinates,
                                        interpolation='bilinear')
```

Define downscaling algorithm

```
In [5]:
downscaled_sm = podpac.algorithm.Arithmetic(A=smap, B=twi, C=twi_bar, D=porosity, E=wilt,
                                            eqn='A + (D - E) / 13.0 * (B - C)')
```

Specify geospatial region and datetime of interests

```
In [6]:
coordinates = podpac.Coordinates([podpac.clinspace(41., 40., 916),
                                  podpac.clinspace(-77, -76, 916),
                                  '2017-09-03T12:00:00'], dims=['lat', 'lon', 'time'])
```

Evaluate downscaling algorithm on local workstation

```
In [7]:
downscaled_soil_moisture = downscaled_sm.eval(coordinates)
```

Evaluate downscaling algorithm via AWS Lambda functions

```
In [8]:
lambda_node = podpac.core.managers.aws_lambda.Lambda(source=downscaled_sm)
downscaled_soil_moisture = lambda_node.eval(coordinates)
```

xarray