

Seamless Transition of Data Analyses and Analytics from a Local Workstation to Scalable, Massively Distributed Processing on the Cloud Using the Open Source PODPAC Library

J. Bieszczad^{*}, M. Ueckermann^{*}, D. Entekhabi[†], D. Callender^{*}, and D. Sullivan^{*} ^{*}Create LLC, [†]Massachusetts Institute of Technology



Motivation

- NASA seeks to **migrate EOSDIS observational data products to the Amazon Web Service (AWS) cloud**
- AWS enables data scientists to **exploit massively scalable processing capabilities** on the cloud
- However, **barriers to entry are high**, given the unfamiliar terminology, setup, workflow, costs, etc. of AWS resources



Project Objectives

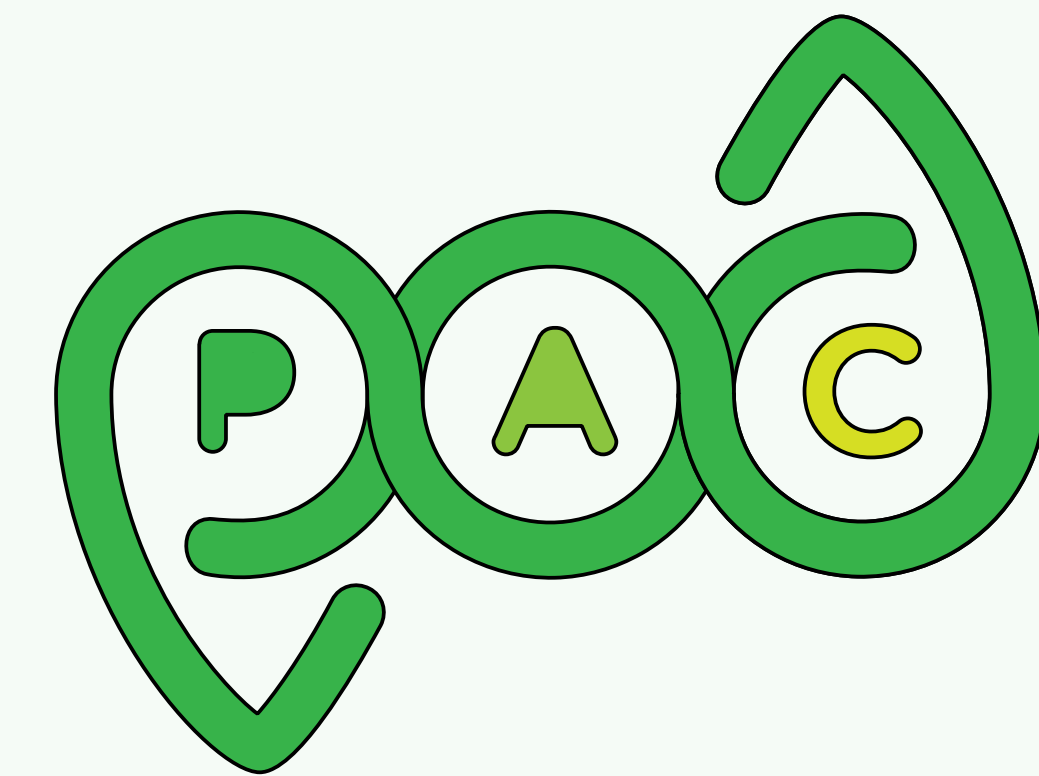
- Develop PODPAC, an open-source, Python library, which **removes major barriers to widespread exploitation of EOSDIS and other earth science data** on the cloud
- Automate geo-data wrangling for **integrated analyses of disparate data sources** in a plug-and-play manner
- Enable data scientists to **easily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

Open Source Development

- PODPAC is 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



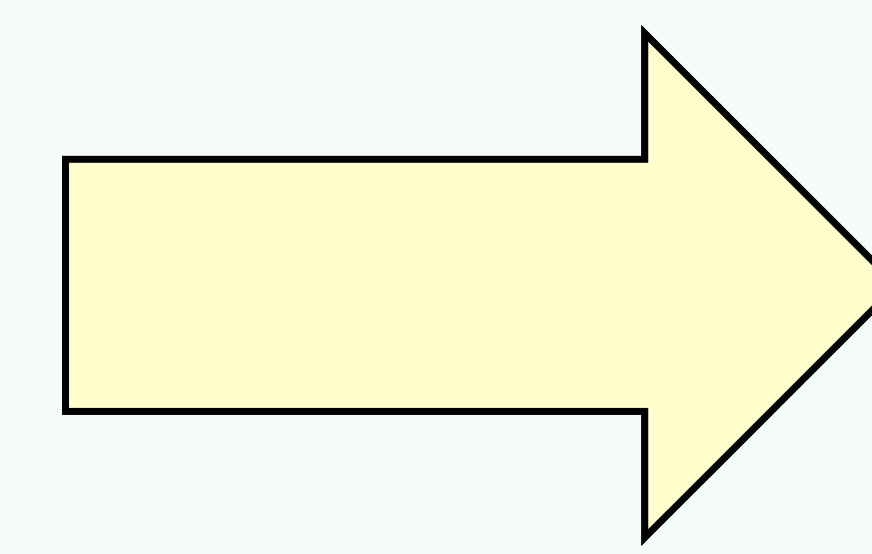
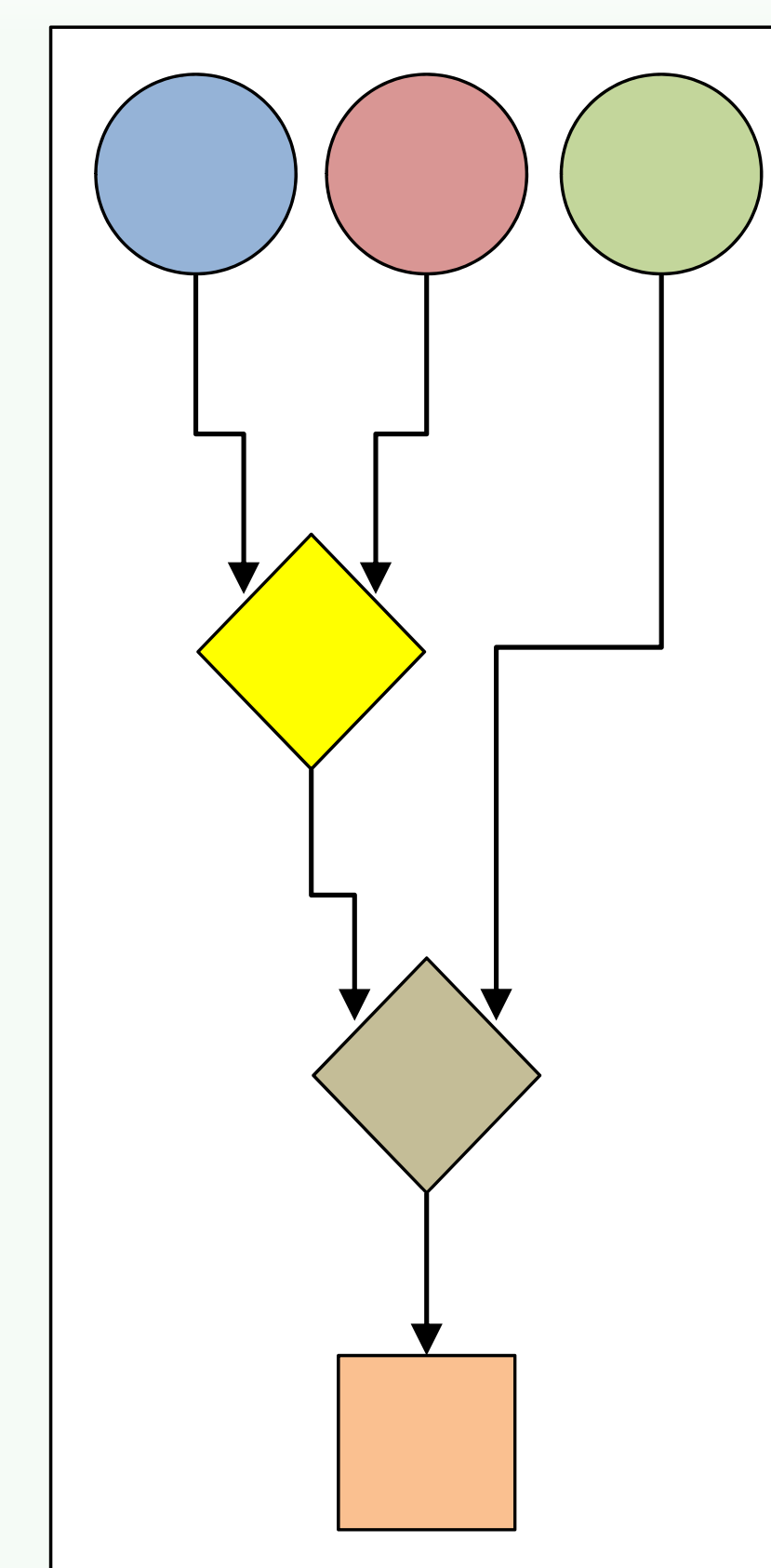
Pipeline for Observational Data Processing, Analysis, and Collaboration

Readily develop integrated geospatial analyses and analytics on your workstation

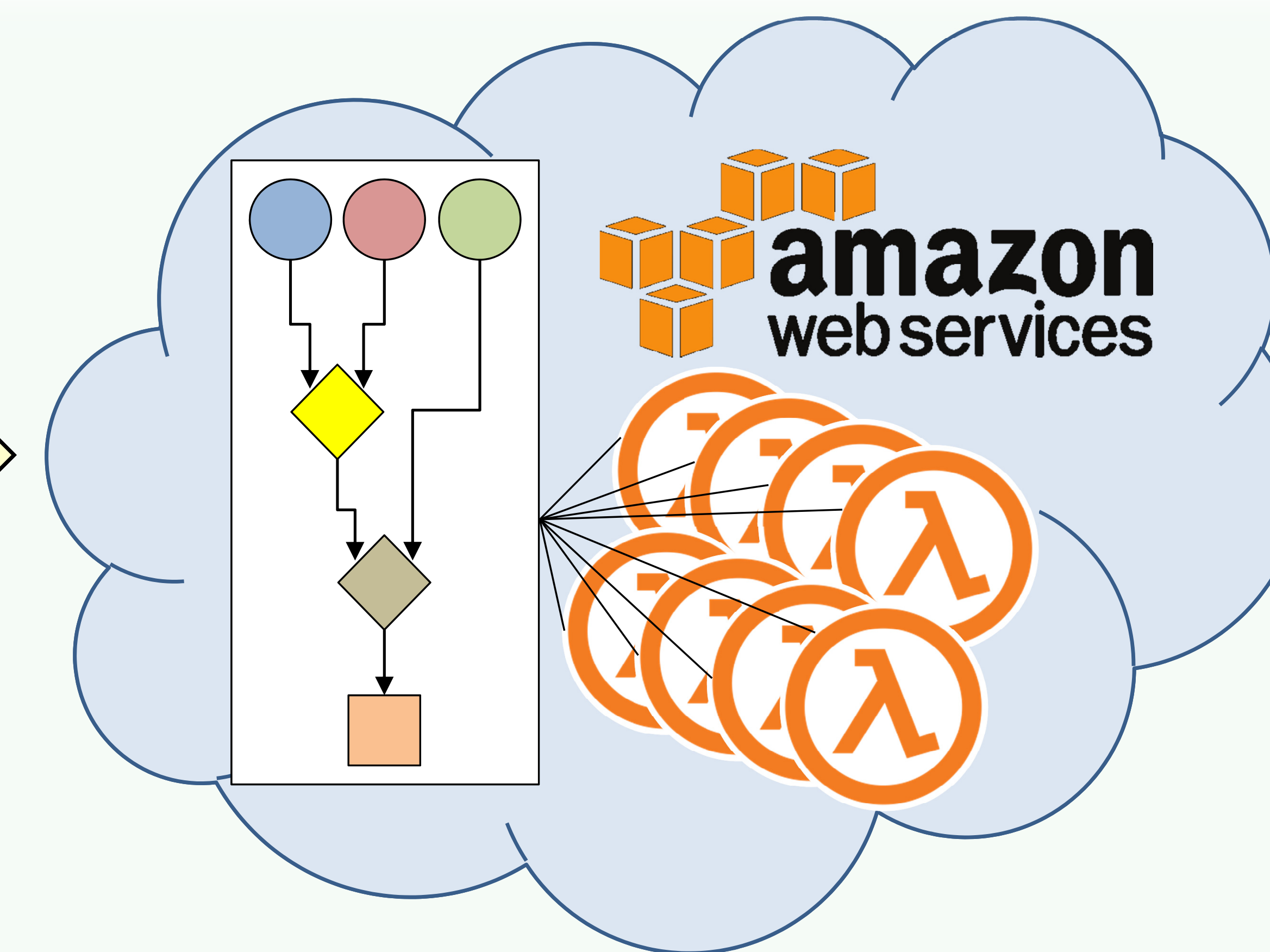
Encapsulated Local and Remote Data Sources

Plug-and-Play User Algorithms

Documented Earth Science Data Products

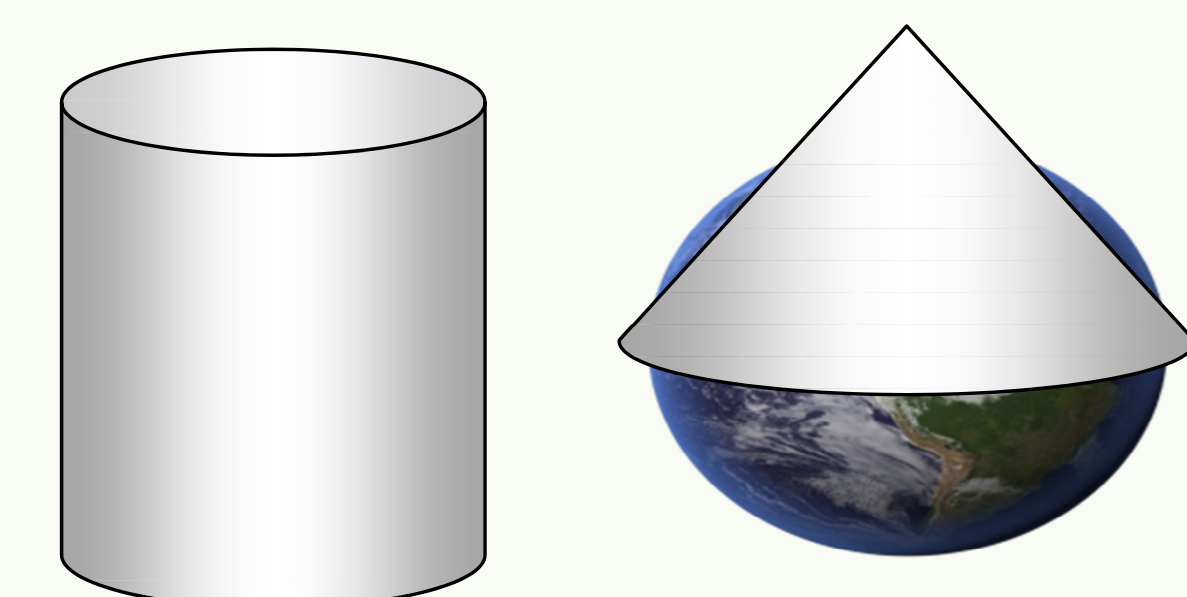


Seamlessly transition to scalable, massively distributed processing on the cloud

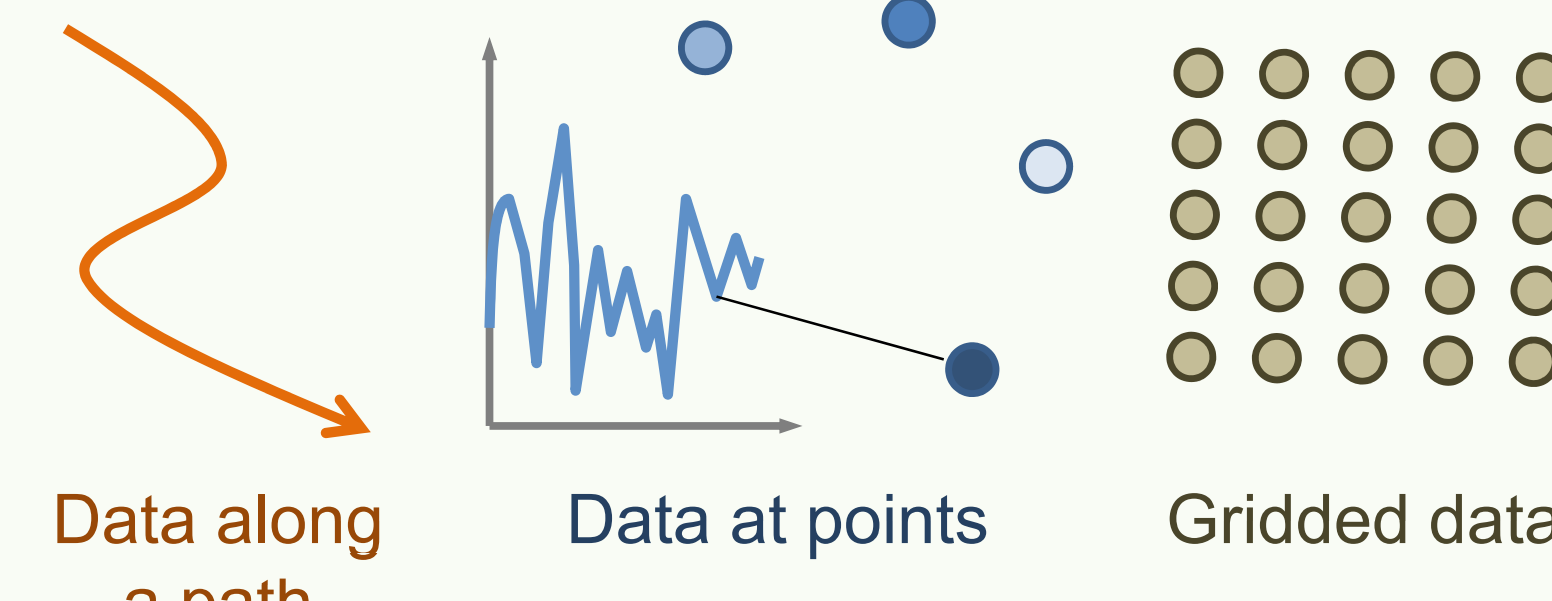


- ✓ Use of Python and Jupyter Notebooks reduces software learning curve for new users
- ✓ Local and remote data (OPeNDAP, WCS, etc.) are encapsulated in common API wrapper for plug-and-play integration within user-specified algorithms
- ✓ Automated data wrangling handles differences in geospatial CRS,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

Geospatial CRS and Projections

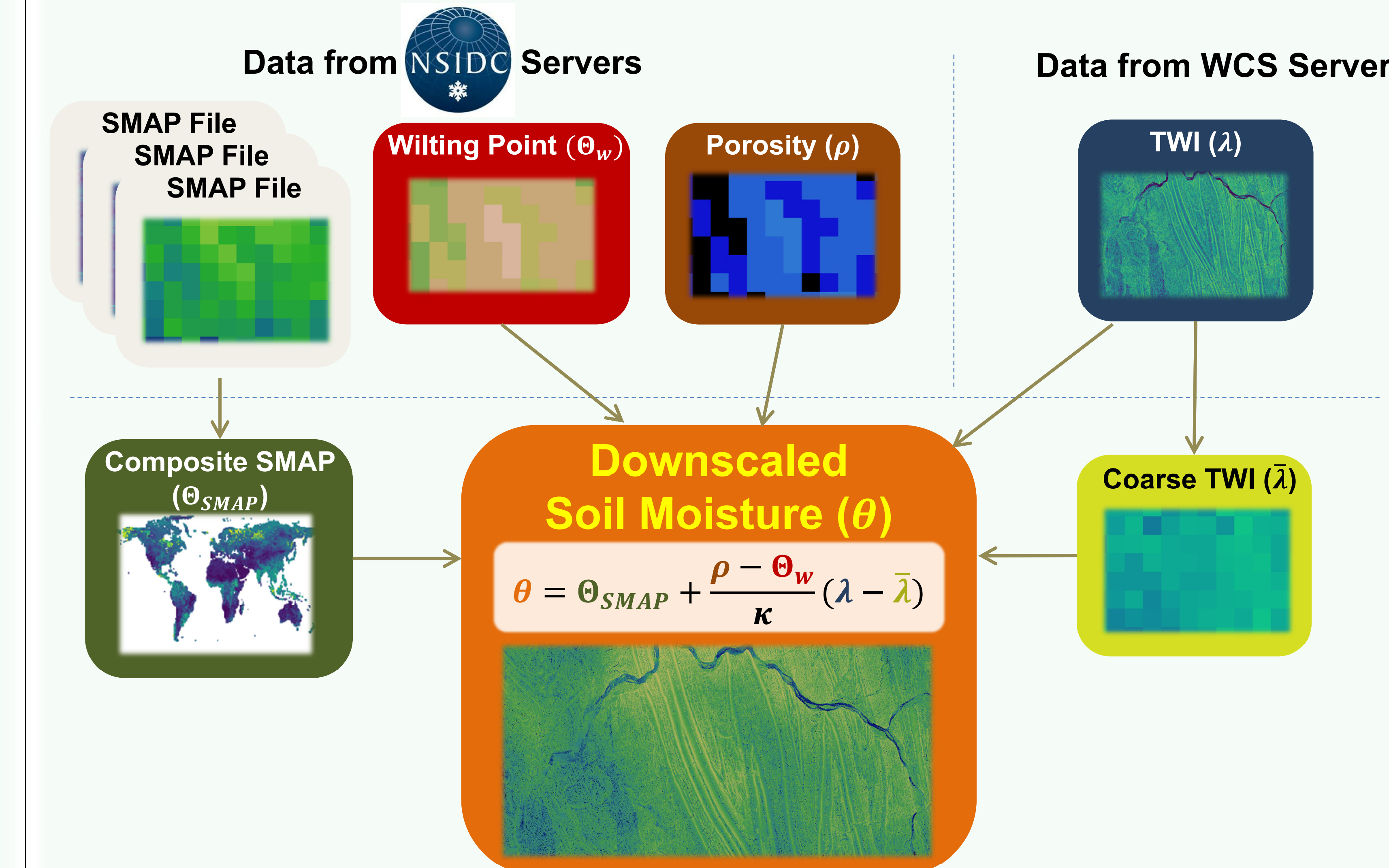


Data Structures



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

“Serverless” SMAP Downscaling



SMAP Downscaling Using PODPAC in a Jupyter Notebook

```
Create PODPAC nodes for accessing NSIDC data via OPeNDAP

In [2]: 1 import podpac # Make the "PODPAC" library available to Python
        2 smap = podpac.datalib.smap.SMAP(interpolation='bilinear', cache_type='disk')
        3 wilt = podpac.datalib.smap.SMAPPWilt(interpolation='bilinear', cache_type='disk')
        4 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]: 1 twi = podpac.data.WCS(source=podpac.utils.load_setting('WCS_URL'),
        2 layer_name=podpac.utils.load_setting('TWI'),
        3 interpolation='nearest')

Reproject high resolution TWI onto low resolution SMAP grid

In [4]: 1 twi_bar = podpac.data.ReprojectedSource(source=twi,
        2 reprojected_coordinates=smap.shared_coordinates,
        3 interpolation='bilinear')

Define downscaling algorithm

In [5]: 1 downscaled_sm = podpac.algorithm.Arithmetic(A=smap, B=twi, C=twi_bar, D=porosity, E=wilt,
        2 eqn='A + (D - E) / 13.0 * (B - C)')

Specify geospatial region and datetime of interests

In [6]: 1 coordinates = podpac.Coordinates([podpac.clinospace(41., 40., 916),
        2 podpac.clinospace(-77., -76., 916),
        3 '2017-09-03T12:00:00'], dims=['lat', 'lon', 'time'])

Evaluate downscaling algorithm on local workstation

In [7]: 1 downscaled_sm.eval(coordinates)

Evaluate downscaling algorithm via AWS Lambda functions

In [8]: 1 lambda_node = podpac.core.managers.aws_lambda.Lambda(source=downscaled_sm)
        2 downscaled_soil_moisture = lambda_node.eval(coordinates)
```

- ✓ 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 NASA’s looming challenges in dealing with massive earth science data volumes

