Xarray: N-D Labeled Arrays and Datasets in Python

Stephan Hoyer (@shoyer)

xarray

Originally (2014-2015) developed at

THE CLIMATE CORPORATION

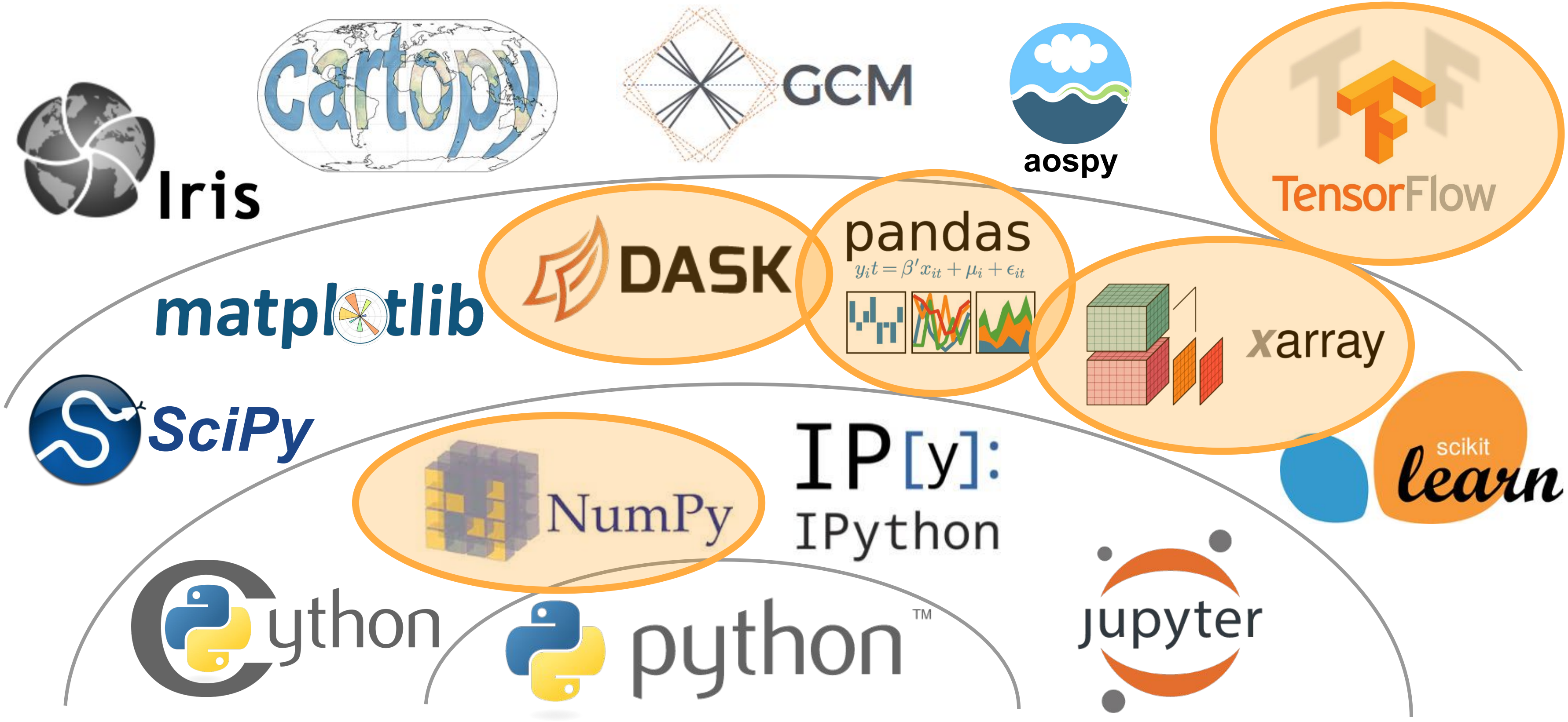
Now, I work at

Google

but this isn't a Google project.

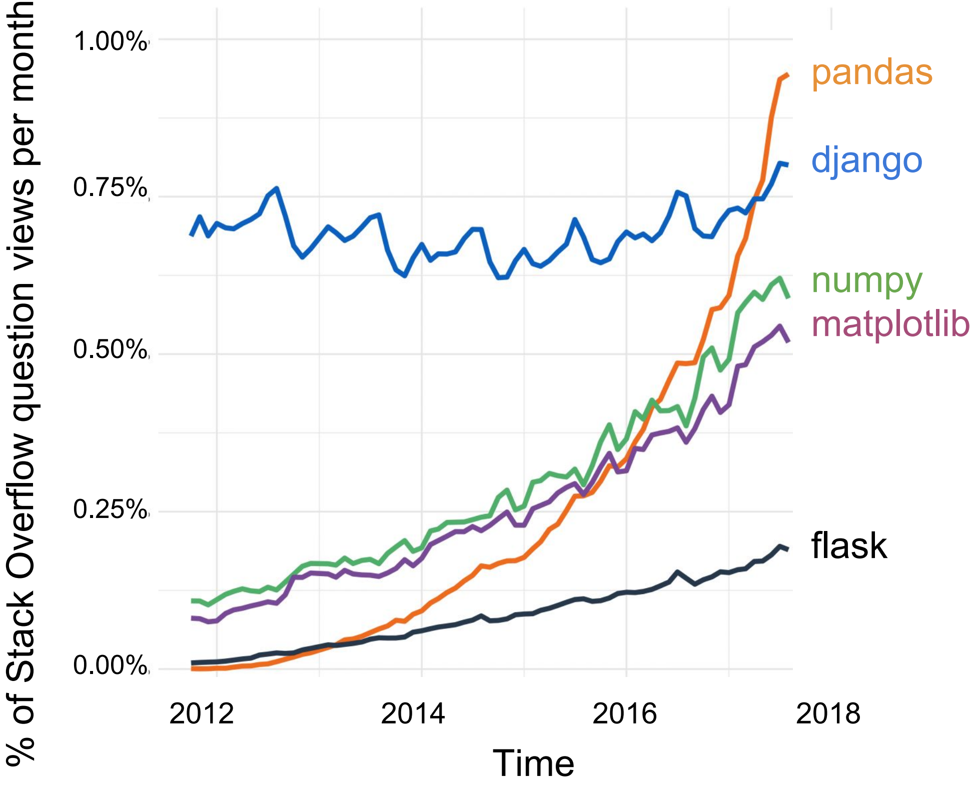
ECMWF Python Workshop, November 28, 2017

Xarray is part of the scientific Python stack



Credit: Jake Vanderplas, SciPy 2015

Why is Python growing so rapidly?



“data science, machine learning and academic research… pandas is the fastest growing Python tag”

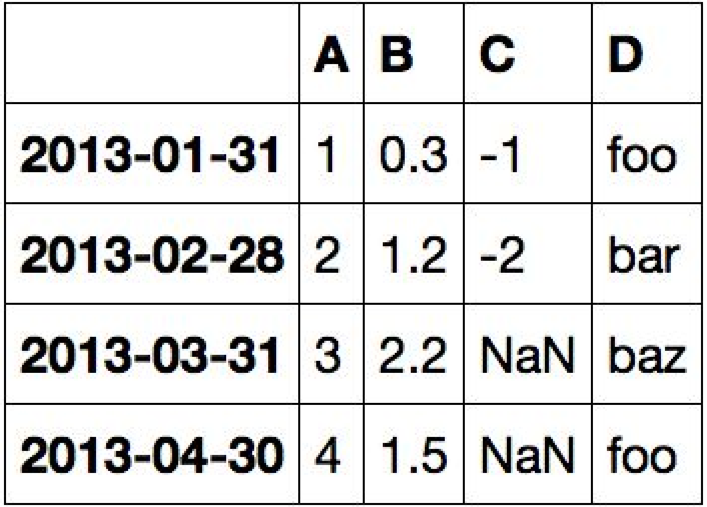
stackoverflow.blog/2017/09/14/
python-growing-quickly

Pandas makes Python data analysis easy

- data frames!
- labels: indexing & alignment
- groupby: split-apply-combine
- missing data
- time series
- plotting
- scipy/pydata stack
- but not N-dimensional

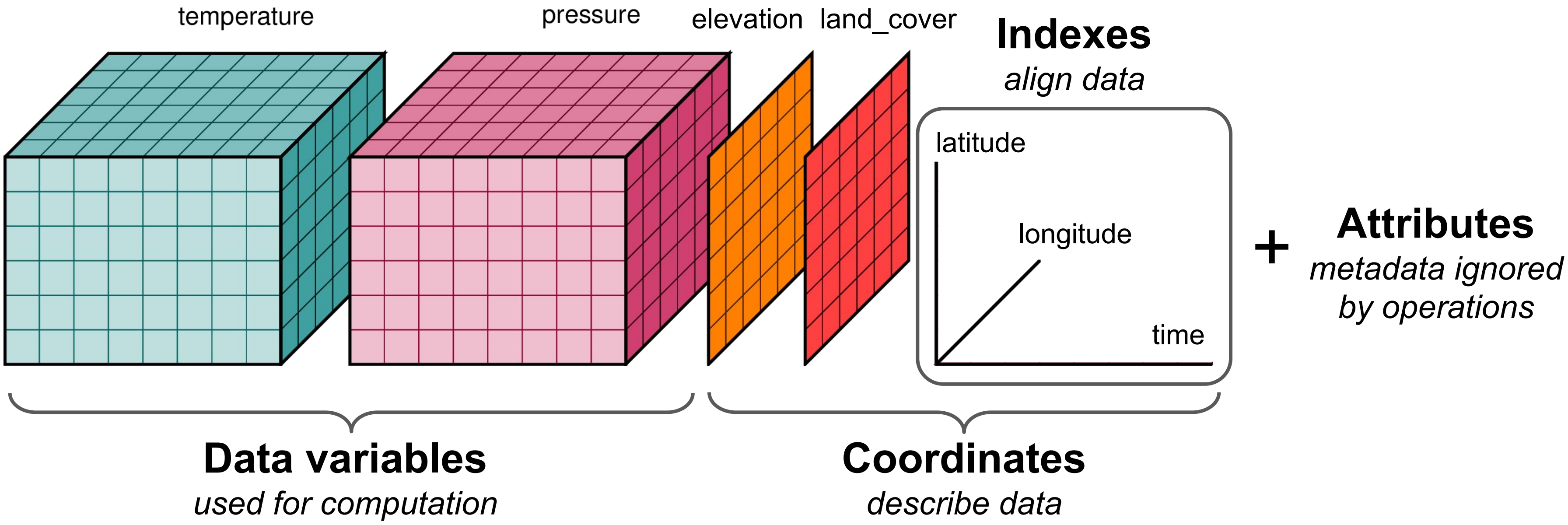
pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



	A	B	C	D
2013-01-31	1	0.3	-1	foo
2013-02-28	2	1.2	-2	bar
2013-03-31	3	2.2	NaN	baz
2013-04-30	4	1.5	NaN	foo

xarray.Dataset: netCDF meets pandas.DataFrame



Indexes
align data

Attributes
metadata ignored by operations

Data variables
used for computation

Coordinates
describe data

Design goals for xarray

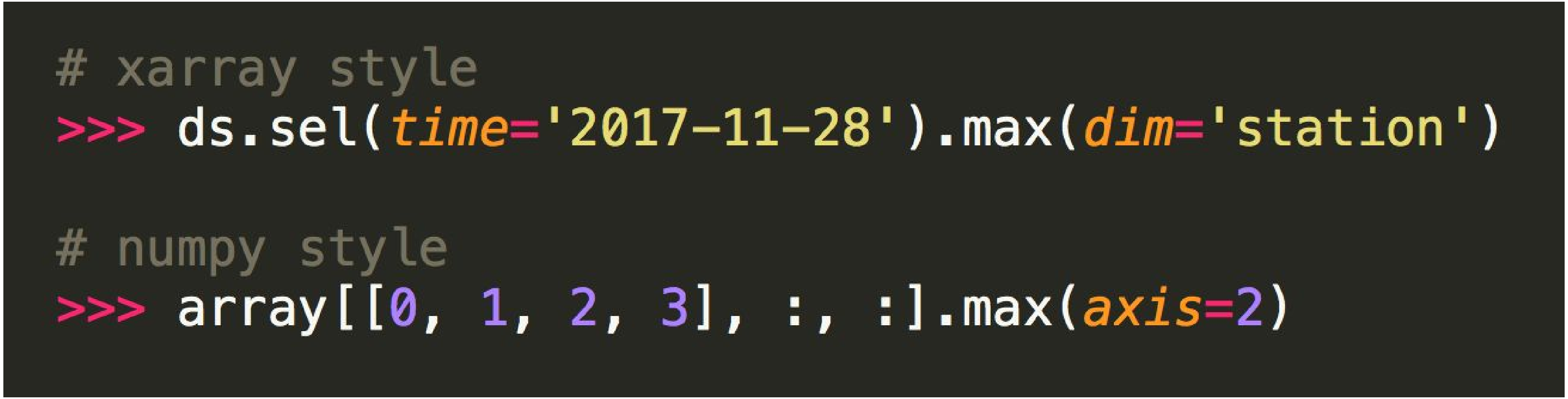
“pandas for N-dimensional arrays”

- build on pandas + NumPy (and now dask)
- copy the pandas API
- use the netCDF data model

Motivated by weather & climate use cases

...but domain agnostic

Xarray operations use names, not numbers



```
# xarray style
>>> ds.sel(time='2017-11-28').max(dim='station')

# numpy style
>>> array[[0, 1, 2, 3], :, :].max(axis=2)
```

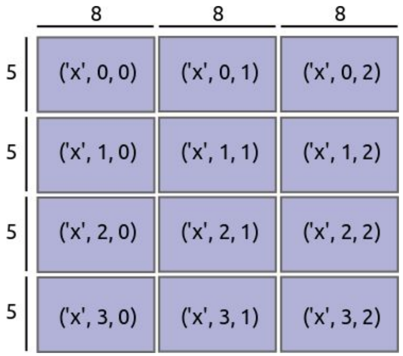
Every operation in xarray is parallelized with Dask

Dask adds two major features to NumPy:

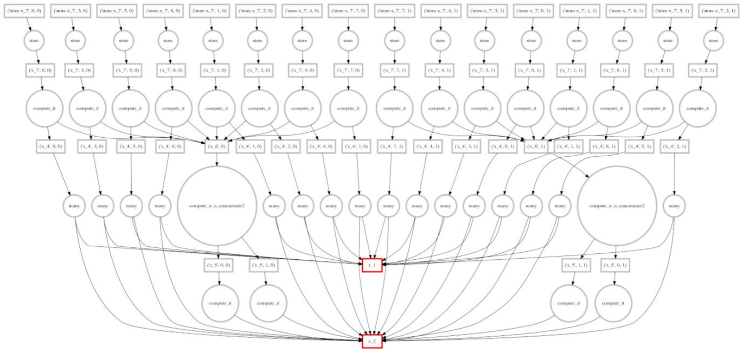
- **Parallelized**: use all your cores
- **Out-of-core**: streaming operations

Dask scales up (to a cluster) *and* down (to a single machine).

To use Dask in xarray, users specify `chunks` or call `open_mfdataset()`.



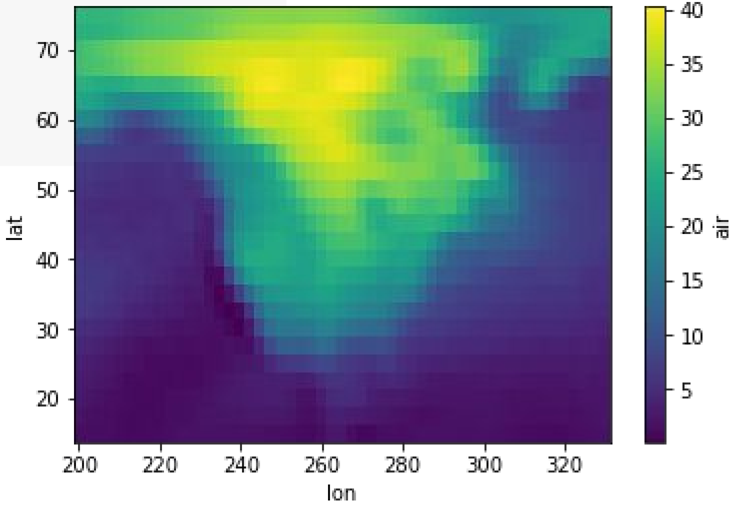
	8	8	8
5	('x', 0, 0)	('x', 0, 1)	('x', 0, 2)
5	('x', 1, 0)	('x', 1, 1)	('x', 1, 2)
5	('x', 2, 0)	('x', 2, 1)	('x', 2, 2)
5	('x', 3, 0)	('x', 3, 1)	('x', 3, 2)



Xarray + Dask makes scalable data analysis easy

```
import xarray

ds = xarray.open_mfdataset('all/your/data/*.nc')
climatology = ds.groupby('time.season').mean('time')
temperature_range = abs(
    climatology.air.sel(season='JJA')
    - climatology.air.sel(season='DJF'))
temperature_range.plot()
```



...but also easily interoperates with the scientific Python stack

Use xarray.apply_ufunc to wrap code for xarray

Handles all the boilerplate involved in wrapping a NumPy function.

Example usage:

```
def spearman_correlation(x, y, dim):
    return xarray.apply_ufunc(
        spearman_correlation_gufunc, x, y,
        input_core_dims=[[dim], [dim]],
        dask='parallelized',
        output_dtypes=[float])
```

Function that supports NumPy style broadcasting

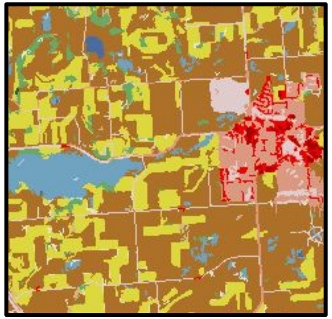
Core dimensions over which the computation takes place

Automatic parallelization with dask!

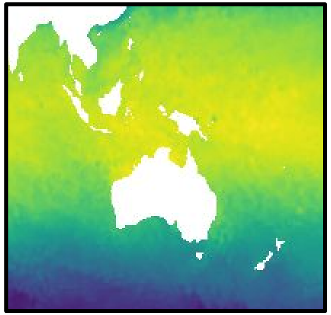
New in xarray v0.10.0

Current data type support in xarray is not enough

Categorical



Missing data



Dates & times

Physical Units

$$52.8\,\mathrm{ft/s} = 36\,\mathrm{mi/h}$$

Two possible solutions:

- NumPy duck arrays: __array_ufunc__ (and __array_concatenate__?)
- Custom NumPy dtypes

Pangeo Data: a community effort for big data geoscience

Domain specific packages building on xarray + dask:

- Data Discovery
- Regions and Shapes
- Regridding
- Signal Processing
- Thermodynamics
- Vector Calculus

pangeo-data.github.io

NSF

COLUMBIA UNIVERSITY

ANACONDA®

NCAR
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Xarray is a community project: join us!

Funded by Pangeo

Stephan Hoyer

Joe Hamman

Ryan Abernathy

Matthew Rocklin

Fabien Maussion

Benoit Bovy

Clark Fitzgerald

Maximilian Roos

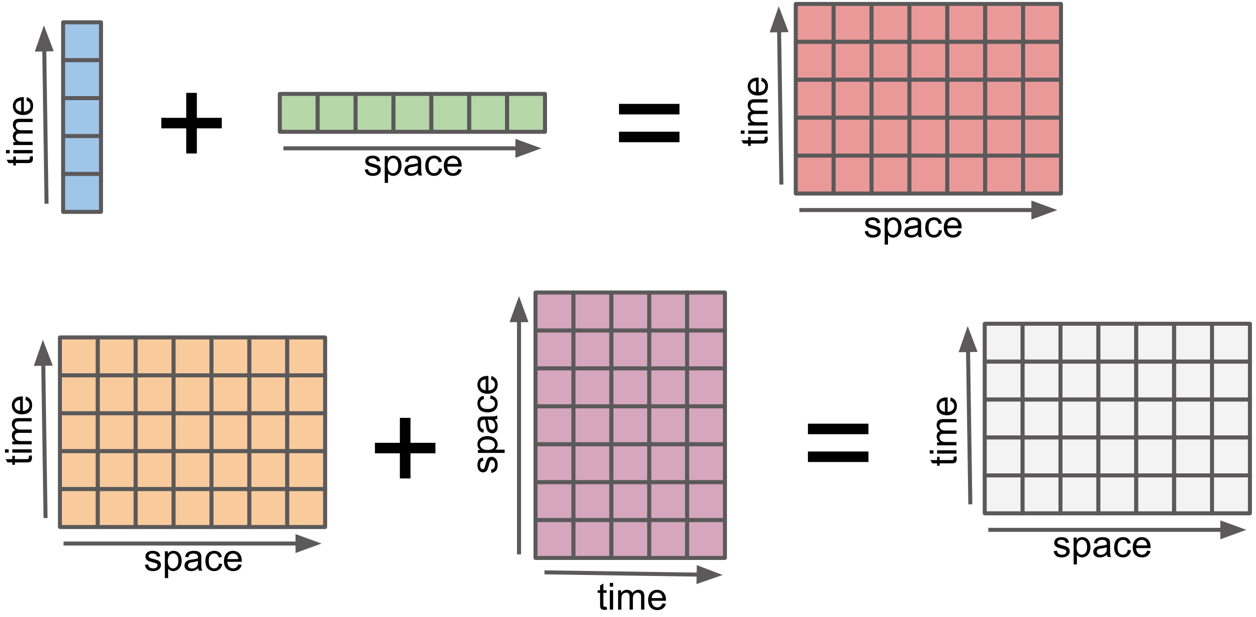
Keisuke Fujii

Not geoscience users!

+ 74 other contributors!

Backup slides

Example: vectorizing by dimension name



Try vectorized indexing! (new in xarray v0.10.0)

Extending xarray with domain specific logic

(1) Composition

```
class MyData:
  def __init__(self):
    self.ds = xr.Dataset()
  …
  def __getitem(self, …):
  …
  def __add__(self, …):
  def __radd__(self, …):
  …
```

Too much work!

(2) Inheritance

```
class MyDataset(
    xarray.Dataset):
  def _merge(self, …):
    super()._merge(…)
```

Too fragile!

(3) Custom accessors

```
@xarray.register_
dataset_accessor(‘my’)
class My:
  …

# later...
ds = xarray.Dataset()
ds.my.custom_method()
```

Just right?