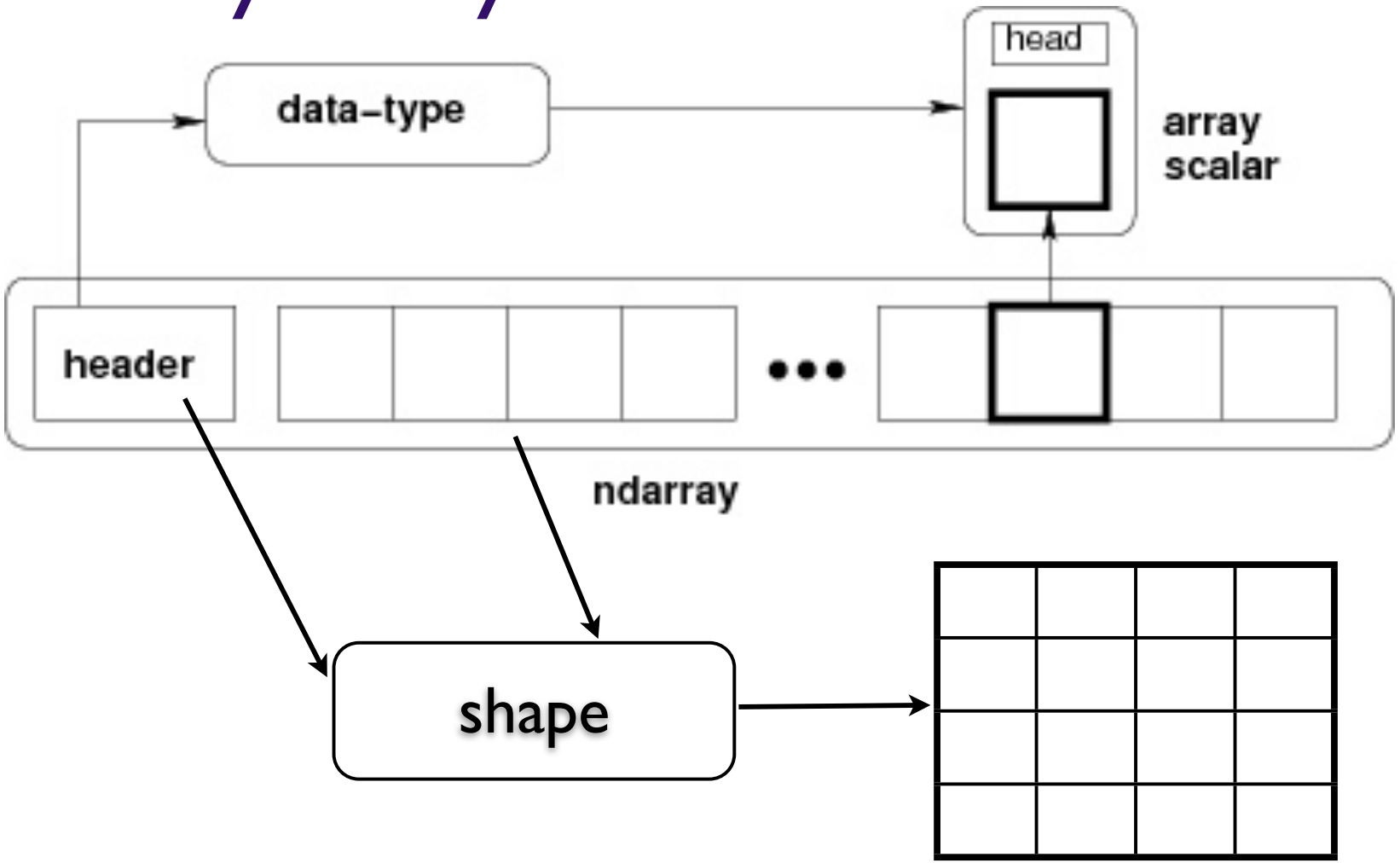
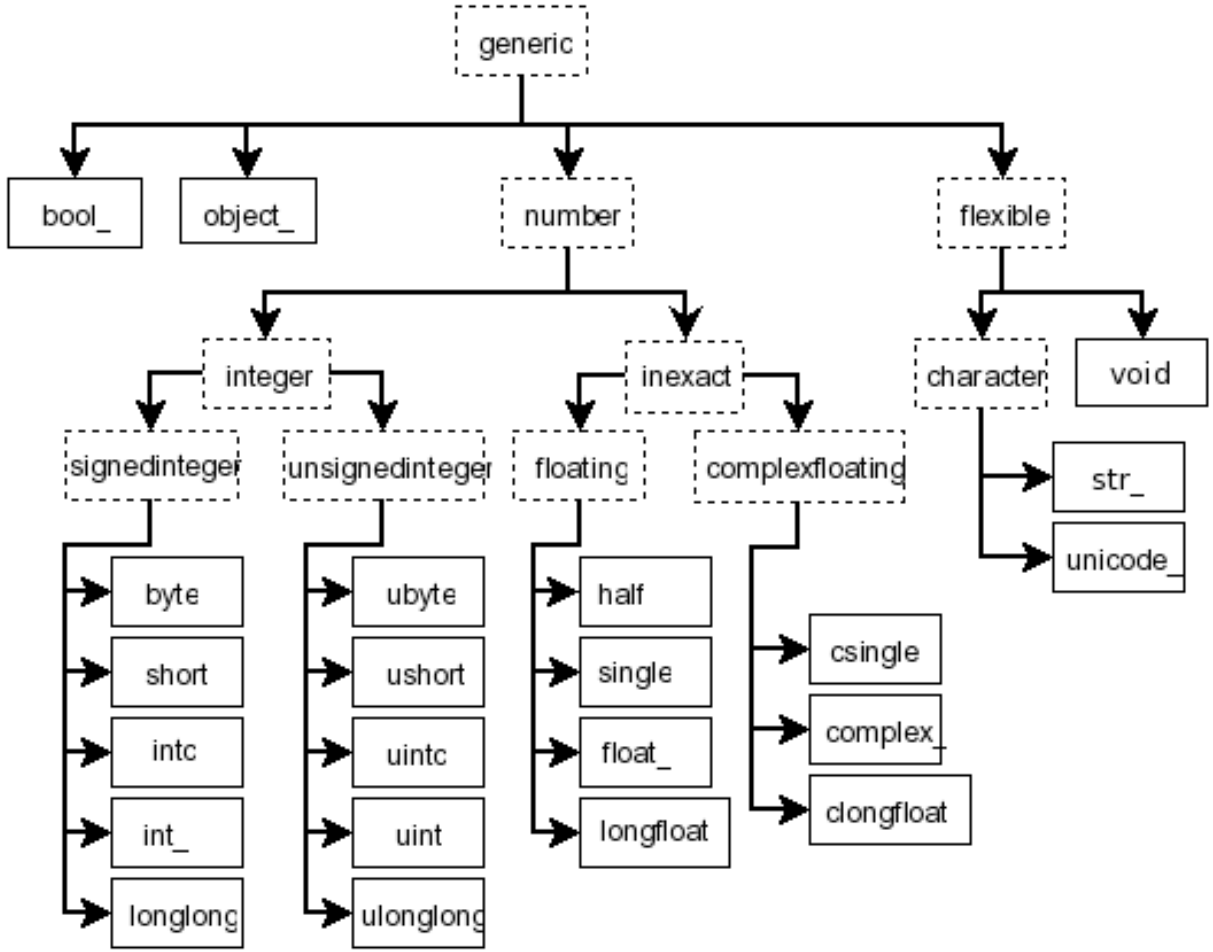
# NumPy Array



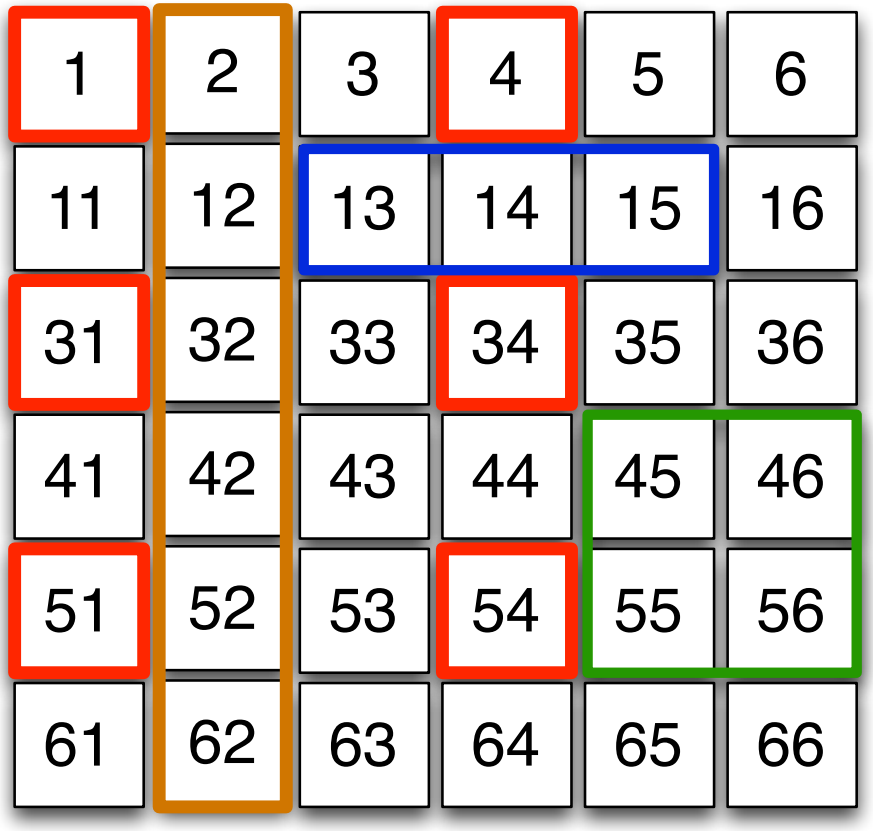
# Broad Type Support




CONTINUUM ANALYTICS

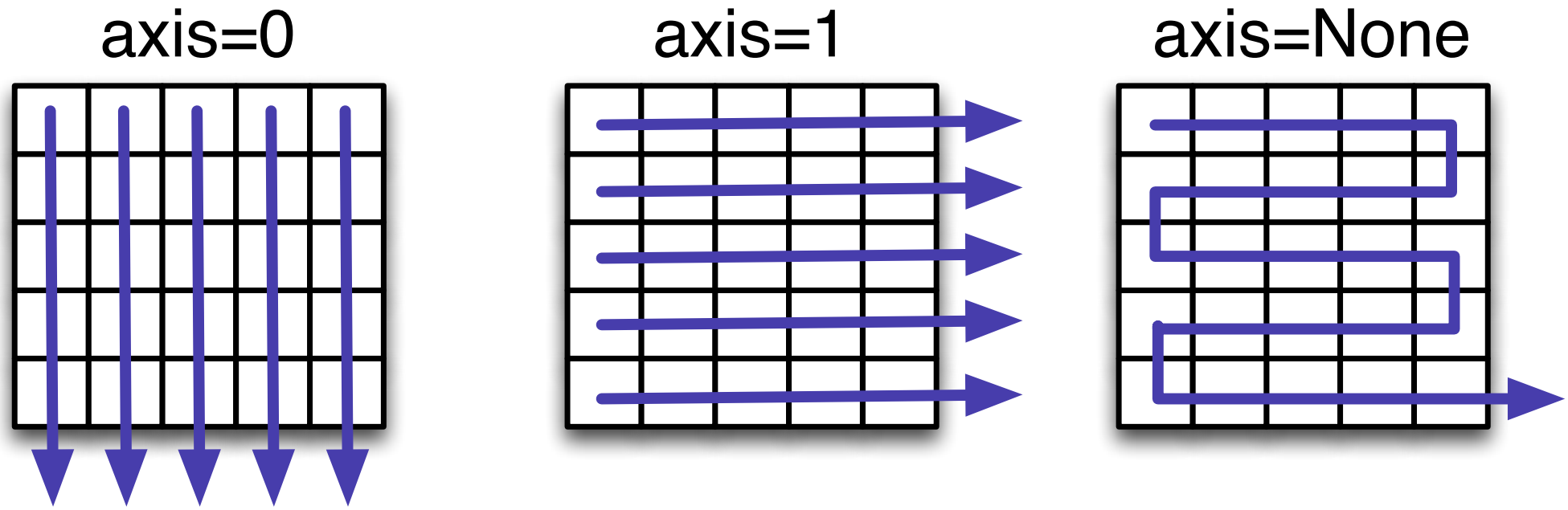
# NumPy Slicing

```
>>> a[:,1]

>>> a[::2,::3]

>>> a[1,2:5]

>>> a[3:5,4:6]
```



# NumPy axis argument (e.g. reduction)

```
a = rand(10,10)
a.std(axis=0)   --- reduce along 0th dimension
a.std(axis=1)   --- reduce along 1st dimension
a.std()         --- reduce along all dimensions
```



# Broadcasting

```
>>> x
array([10, 20])
>>> x.shape
(2,)
```

```
>>> x[np.newaxis,:]
array([[10, 20]])
>>> x[np.newaxis,:].shape
(1, 2)
```

```
>>> y
array([[1],
       [2],
       [3]])
>>> y.shape
(3, 1)
```

```
>>> x+y
array([[11, 21],
       [12, 22],
       [13, 23]])
>>> (x+y).shape
(3, 2)
```

# Zen of NumPy

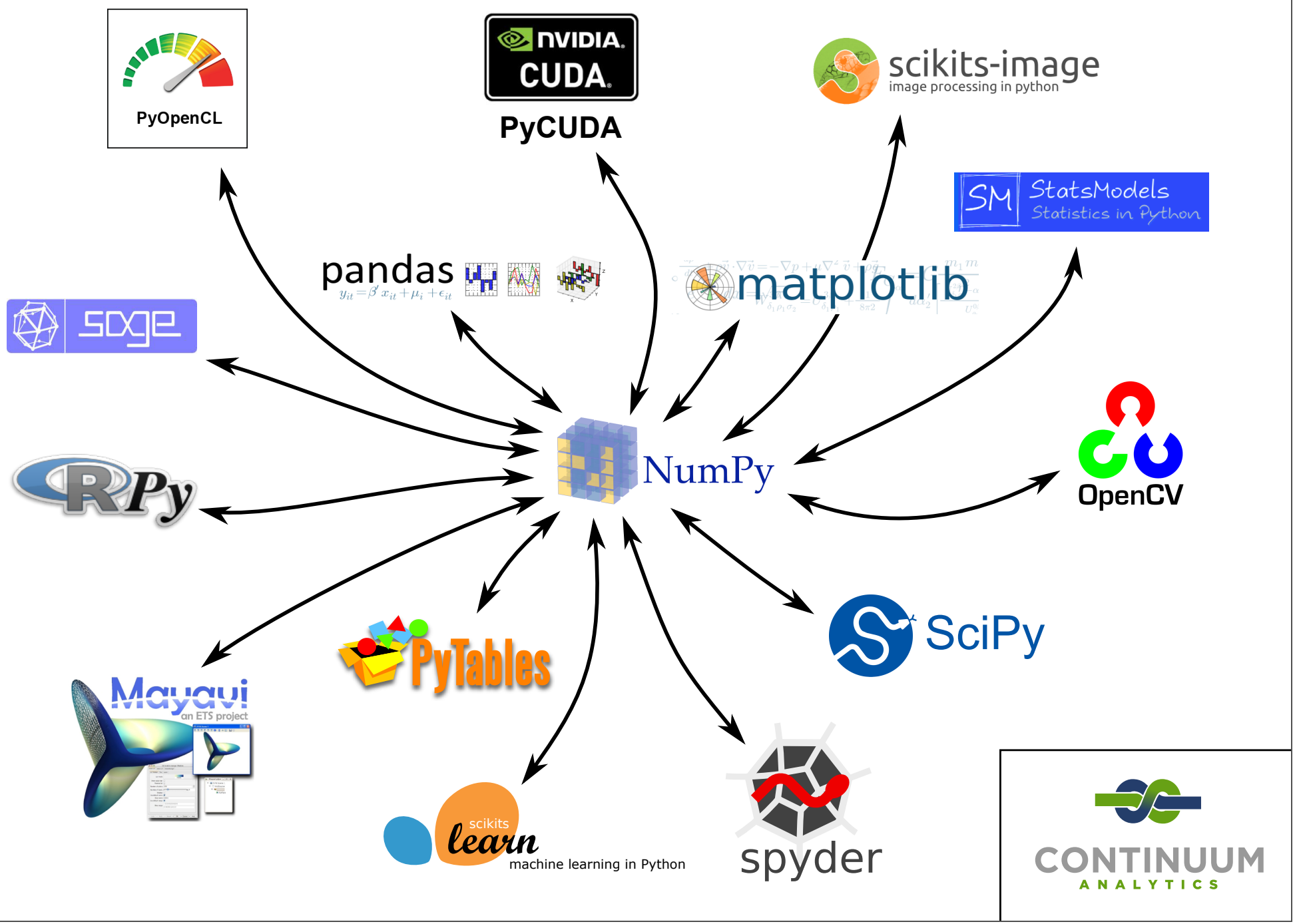
- strided is better than scattered
- contiguous is better than strided
- descriptive is better than imperative
- array-oriented is better than object-oriented
- broadcasting is a great idea
- vectorized is better than an explicit loop
- unless it's too complicated --- then use Numba
- think in higher dimensions

# Benefits of Array-oriented

- Many technical problems are naturally array-oriented (easy to vectorize)
- Algorithms can be expressed at a high-level
- These algorithms can be parallelized more simply (quite often much information is lost in the translation to typical "compiled" languages)
- Array-oriented algorithms map to modern hard-ware caches and pipelines.

# What is good about NumPy?

- Array-oriented:  slicing and broadcasting
- Extensive Dtype System (including structures)
- C-API
- Simple to understand data-structure
- Memory mapping
- Syntax support from Python
- Large community of users and packages that build on it
- Easy to interface C/C++/Fortran code


PyOpenCL
NVIDIA CUDA
PyCUDA
scikits-image
image processing in python
StatsModels
Statistics in Python
pandas
matplotlib
sage
NumPy
RPy
OpenCV
PyTables
SciPy
Mayavi
an ETS project
scikits
learn
machine learning in Python
spyder
CONTINUUM
ANALYTICS