DistArray: from Numpy to parallel computing, seamlessly

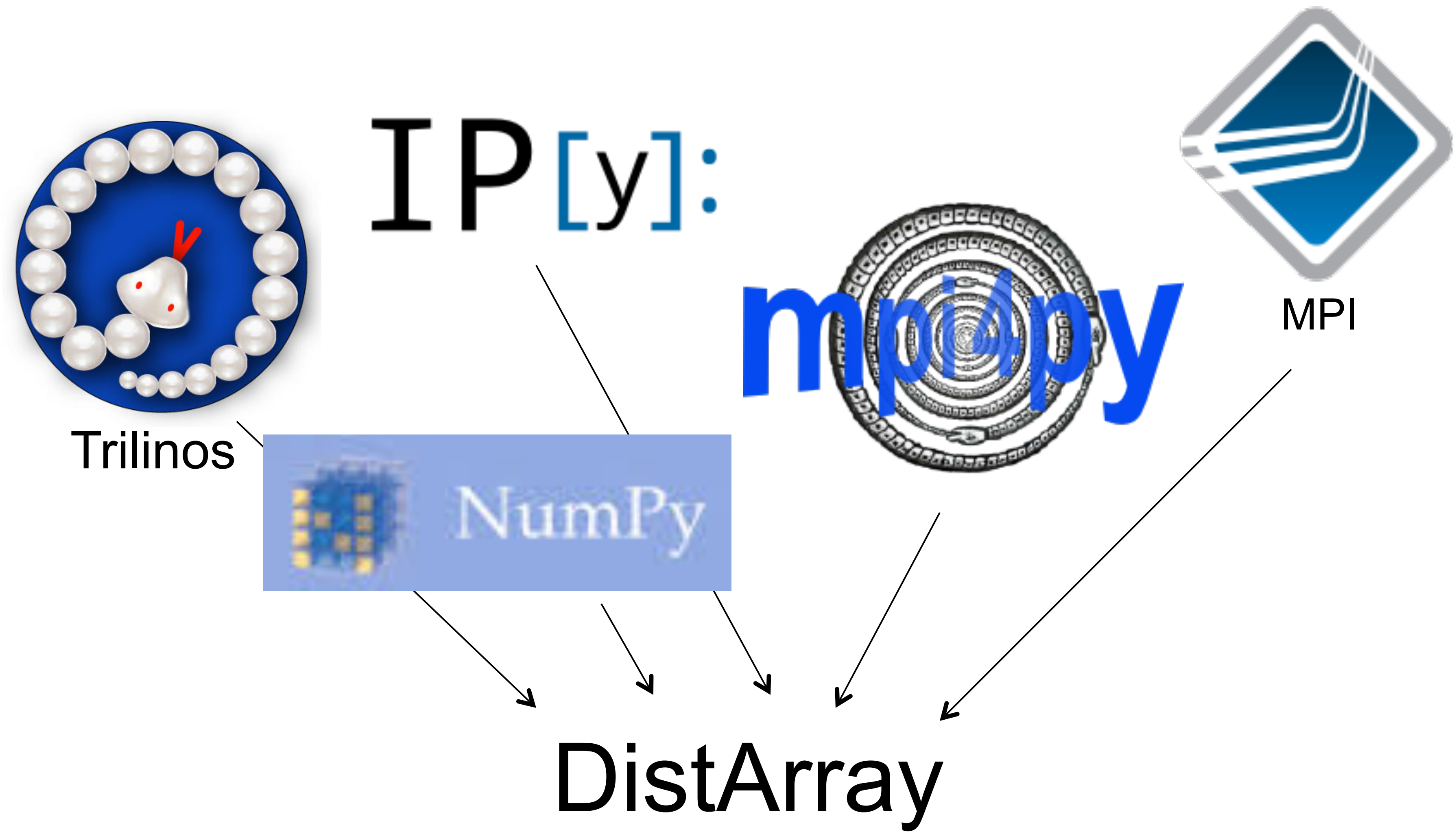
Jonathan Rocher
jrocher@enthought.com

A few questions

- How many people are dealing with **large amounts of data**?
- How many people like python because it is **simple to write**?
- How many people like IPython because it is **interactive**?
- How many people already know **Numpy**?
- How many people have **multiple cores** on their machines? Access to a **cluster**?
- How many companies/labs have a **cluster that isn't used** a lot by scientists?

Why should you give up **interactive python** when you want to **parallelize your programs**?

What is the DistArray project?



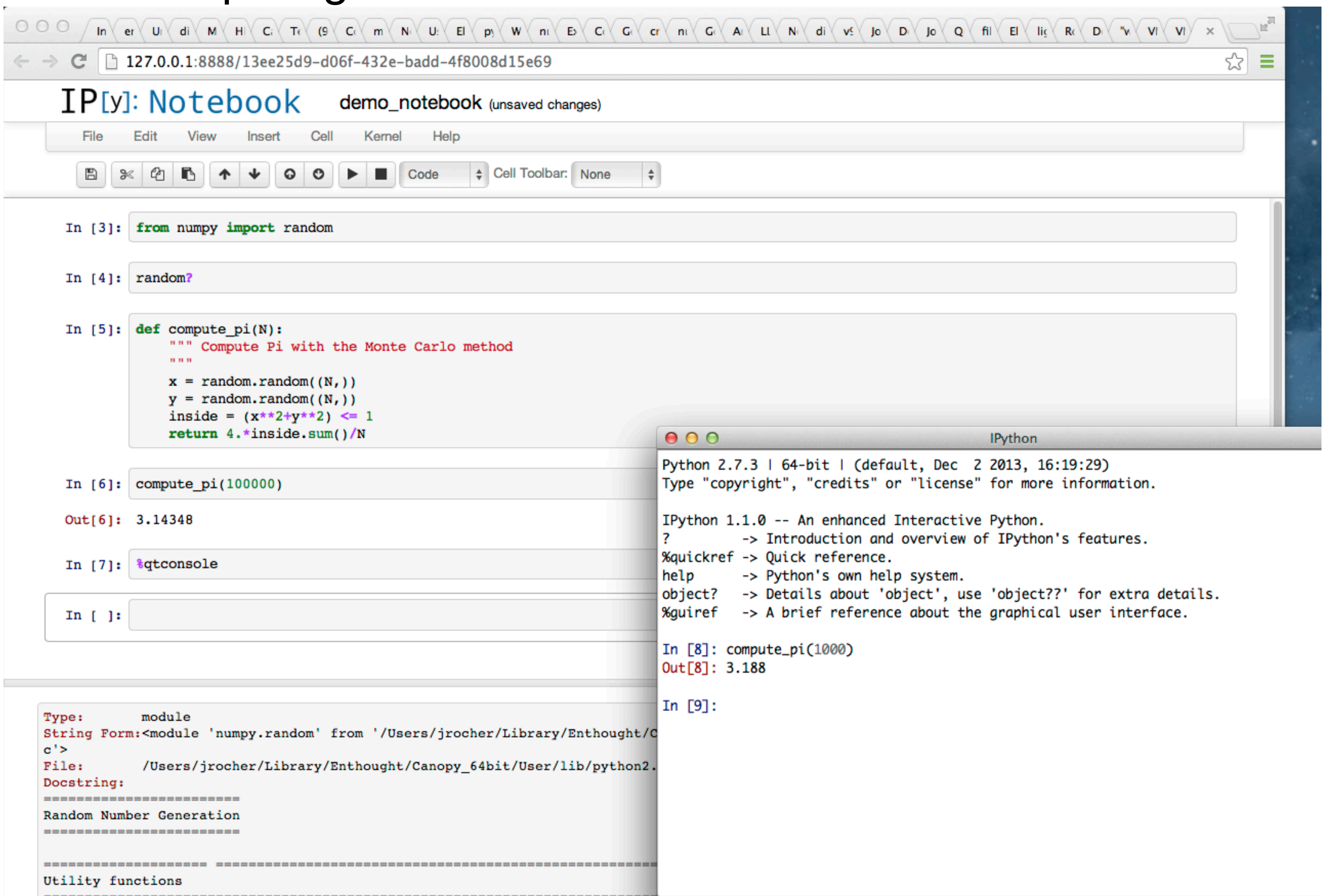
What is the DistArray project?

- SBIR funded **open source** project,
- developed at **Enthought** by a team led by Kurt Smith,
- partnering with Bill Spotz from **Sandia**'s `(py)Trilinos` project, and Brian Granger from the **`IPython`** team,
- to go **seamlessly** from a `NumPy` processing function to a **parallel** program on multiple cores/CPUs/nodes in a cluster.
- Targets users who
    - need more than 1 node but less than 10^3
    - have a lot of data, maybe already distributed
    - need **parallel computation without loosing comfort of Python**

- Warning: still in its infancy!

What is IPython?

- Interactive console for python interpreter
- A parallel computing infrastructure



What is Numpy?

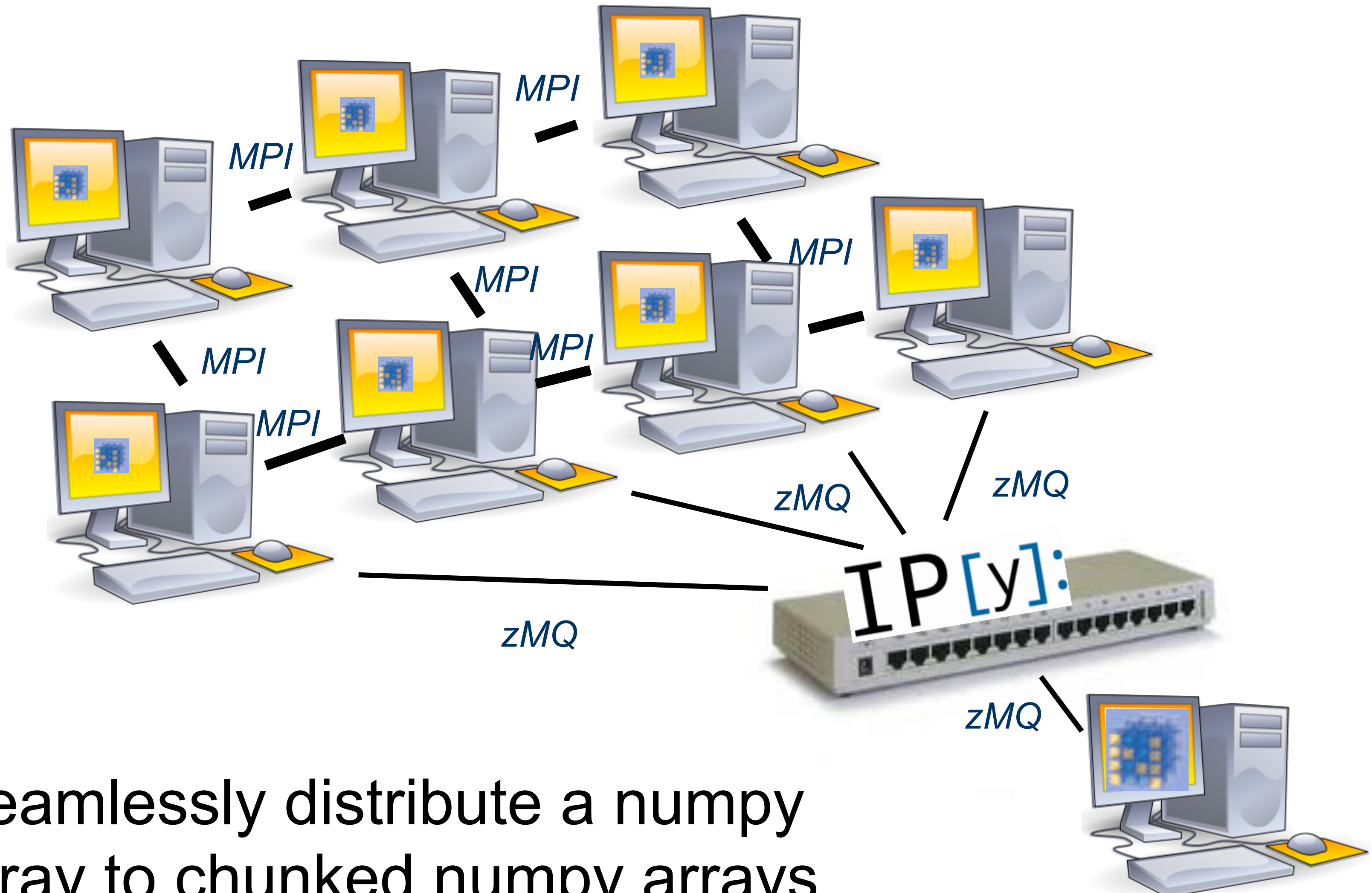
- An **array based computation** package for python, written in C
- The simplest way to do efficient computations in python. You get numpy arrays from any file loader like netCDF's, gridapi, HDF5, ...

```
>>> from numpy import *
>>> x = linspace(0, 2*pi, 10)
>>> y = sin(x)
>>> y > 0.
>>> z = random.random((100, 100))
>>> filter = array([[0, 0, 1],
                    [0, 1, 0],
                    [1, 0, 0]])
>>> from scipy.signal import convolve
>>> convolve(z, filter)
```

Numpy + IPython.parallel + MPI

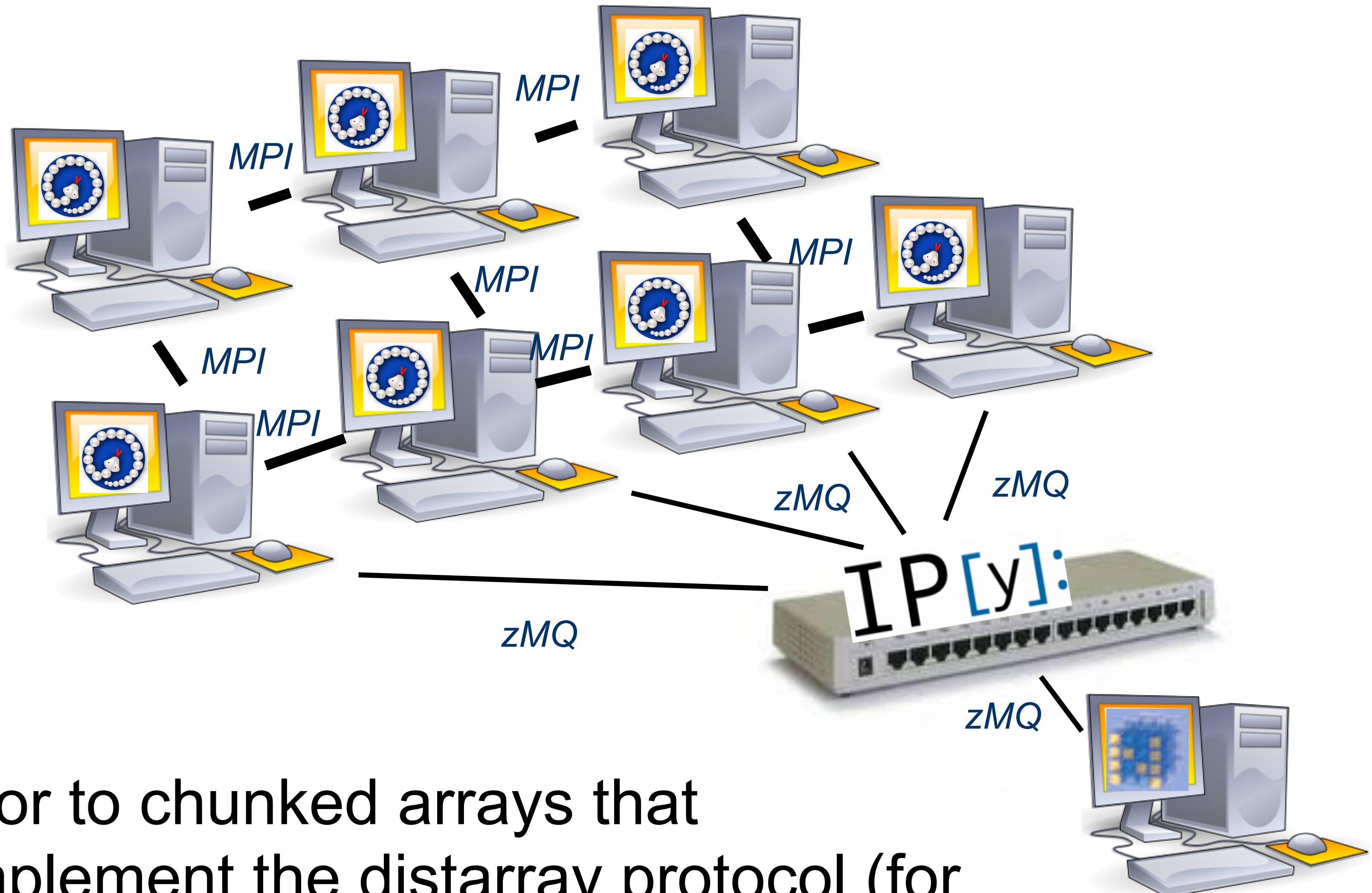
- Assumes that you are dealing with so much data that it cannot fit inside 1 machine.
- Assumes that you already know vector based computations with Numpy.
- Assumes that you don't have time to learn C++, MPI, Trilinos, ...
- Defines a **distributed array protocol** that parallel libraries can understand. That protocol is developed with and adopted by Sandia's `PyTrilinos` and PNNL's `GlobalArray` projects. More to come...
- A client node (your laptop?) connects to a local/remote cluster of engines and dispatches jobs without data transfer: data is created or loaded on the nodes.
- Makes **embarrassingly parallel** problems even more embarrassing and will supports **inter-node MPI communication** for the others.
- Allows fine grained control of distribution mechanism: 'block', 'cyclic', 'block-padded', 'cyclic-block', 'unstructured', or 'not distributed'. Each dimension can be distributed differently.

Numpy + IPython.parallel + MPI



Seamlessly distribute a numpy array to chunked numpy arrays...

Numpy + IPython.parallel + MPI



...or to chunked arrays that implement the distarray protocol (for e.g. GA or pyTrilinos).

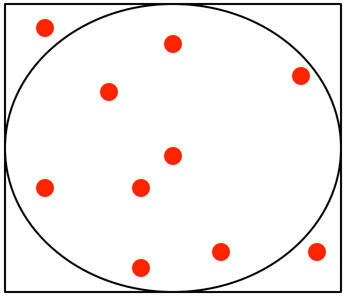
Distributed array operations

At this point, a "distributed toolbox" is available with mathematical operations:

```
>>> from distarray import dist_numpy
>>> dir(dist_numpy)
['absolute', 'add', 'arccos', 'arccosh', 'arcsin', 'arcsinh',
'arctan', 'arctan2', 'arctanh', 'bitwise_and', 'bitwise_or',
'bitwise_xor', 'conjugate', 'cos', 'cosh', 'divide', 'empty',
'exp', 'expm1', 'floor_divide', 'fmod', 'fromarray',
'fromfunction', 'fromndarray', 'hypot', 'invert',
'left_shift', 'log', 'log10', 'log1p', 'multiply',
'negative', 'ones', 'power', 'reciprocal', 'remainder',
'right_shift', 'rint', 'sign', 'sin', 'sinh', 'sqrt',
'square', 'subtract', 'tan', 'tanh', 'target_to_rank',
'targets', 'true_divide', 'view', 'zeros']
>>> distributed_numpy.ones((300, 10))
<DistArray(shape=(300, 10), targets=[0, 1, 2, 3])>
>>> _.get_localshapes()
[(75, 10), (75, 10), (75, 10), (75, 10)]
```

Monte Carlo estimate for π

Throw darts in a square: the number of them inside the circle give you an estimate of π.



```
import numpy

def estimate_pi(N):
    """ Compute an estimation of pi using the classic Monte Carlo
    method.

    Parameters:
    -----------
    N : Number of trial for the MonteCarlo
    """
    x = numpy.random.random(N)
    y = numpy.random.random(N)
    inside = numpy.hypot(x, y) <= 1
    num_inside = inside.sum()
    return 4. * num_inside / N

pi = estimate_pi(1e4)
```

Parallel MC estimate for π

```
from distarray import dist_numpy

def estimate_pi(N):
    """ Compute an estimation of pi using the classic Monte Carlo
    method. Create the data on each node block-distributed.

    Parameters:
    -----------
    N : Number of trial for the MonteCarlo
    """
    x = dist_numpy.random.rand(N)
    y = dist_numpy.random.rand(N)
    inside = dist_numpy.hypot(x, y) <= 1.
    num_inside = inside.sum()
    return 4. * n_inside/n

pi = estimate_pi(1e4)
```

Another parallel MC estimate for π

```
from distarray.client import Context
from distarray import odin
context = Context()

@odin.local
def pi_montecarlo(n):
    """Get an estimation of pi on each engine."""
    import numpy
    x = numpy.random.rand(n)
    y = numpy.random.rand(n)
    inside = numpy.hypot(x, y)
    return 4*numpy.sum(r <= 1)/float(n)

N_on_each_engine = 1e4/len(context.view)
pi_estimates = pi_montecarlo(N_on_each_engine)
```

Roadmap

Version 0.2 (Apr 2014): Minimum viable product

- Basic communication, mathematical operations
- `local` decorator
- Export/Import with PyTrilinos

Version 0.3 (Apr 2015): Public release

- Slicing, broadcasting?, fancy indexing?
- Distributed IO operations (chunked txt, chunked bin, HDF5)
- Redistribution
- Expression analysis for “latency hiding”

Version 1.0:

- Integrated inside IPython
- A lot more stuff!

More details? Want to help?

- Want to **contribute**?

check out `distarray`'s public repository:
github.com/enthought/distarray

- Want to leverage this effort for **your implementation** of a distributed array?

https://github.com/enthought/distributed-array-protocol

- Want to **partner with us** or support the development toward your needs?

Contact us at info@enthought.com