Arkouda

αρκούδα

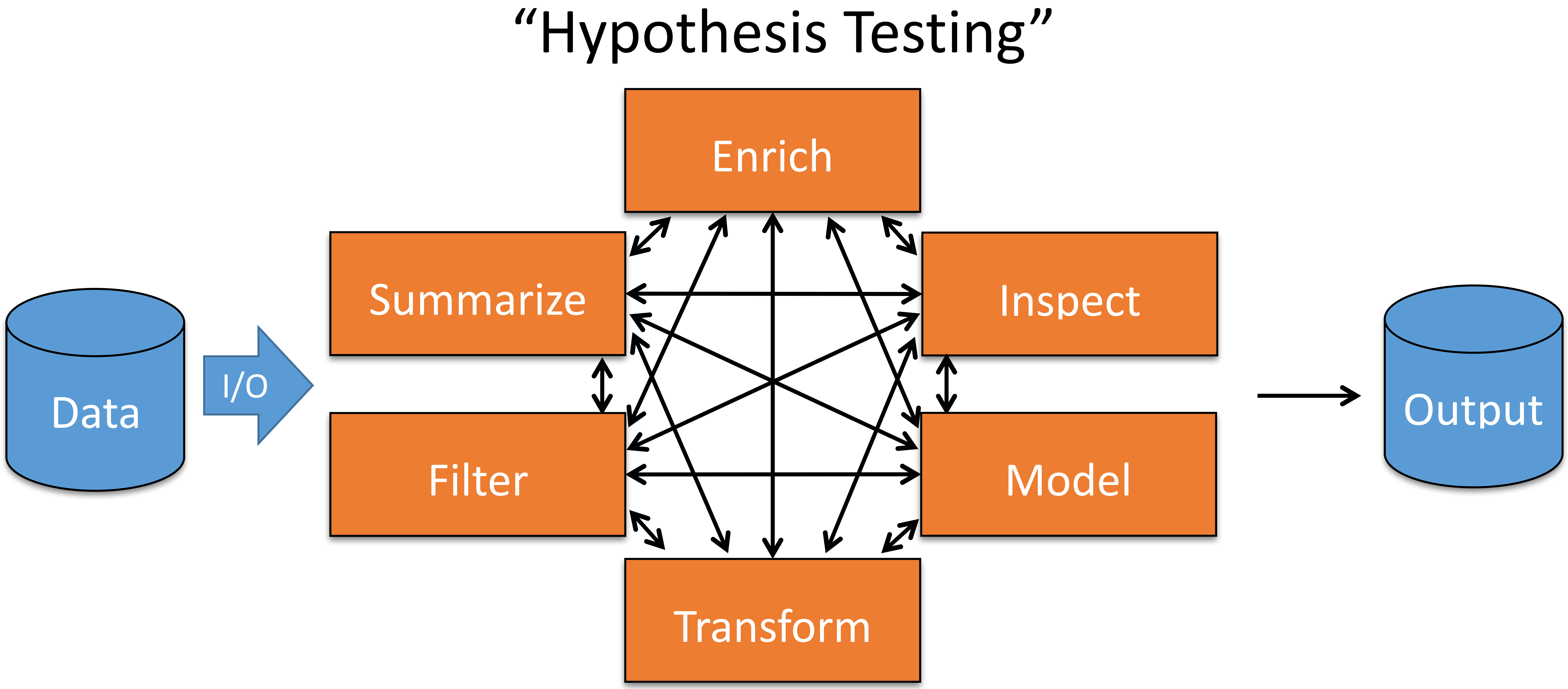
NumPy-like arrays at massive scale
backed by Chapel.

Michael Merrill (presenting)
William Reus
Timothy Neumann
PAW-ATM 2019
November 17, 2019

We want some
of our
Data
Scientists
to drive
an F22!

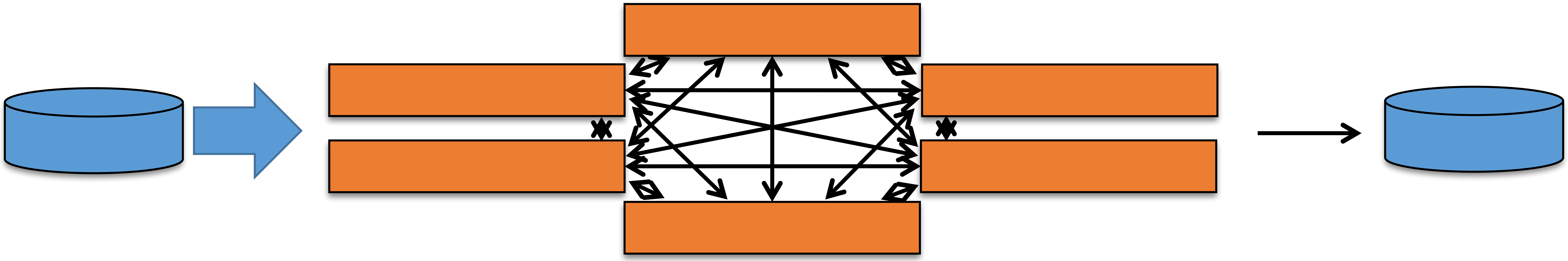
Jupyter allows
Data
Scientists
to drive a
cool plane!

Why HPC enabled EDA?



We want to do EDA on 10s to 100s of terabytes…
In Data Science everyone talks about AI/ML, those things can only come from EDA!

Implications for Computing

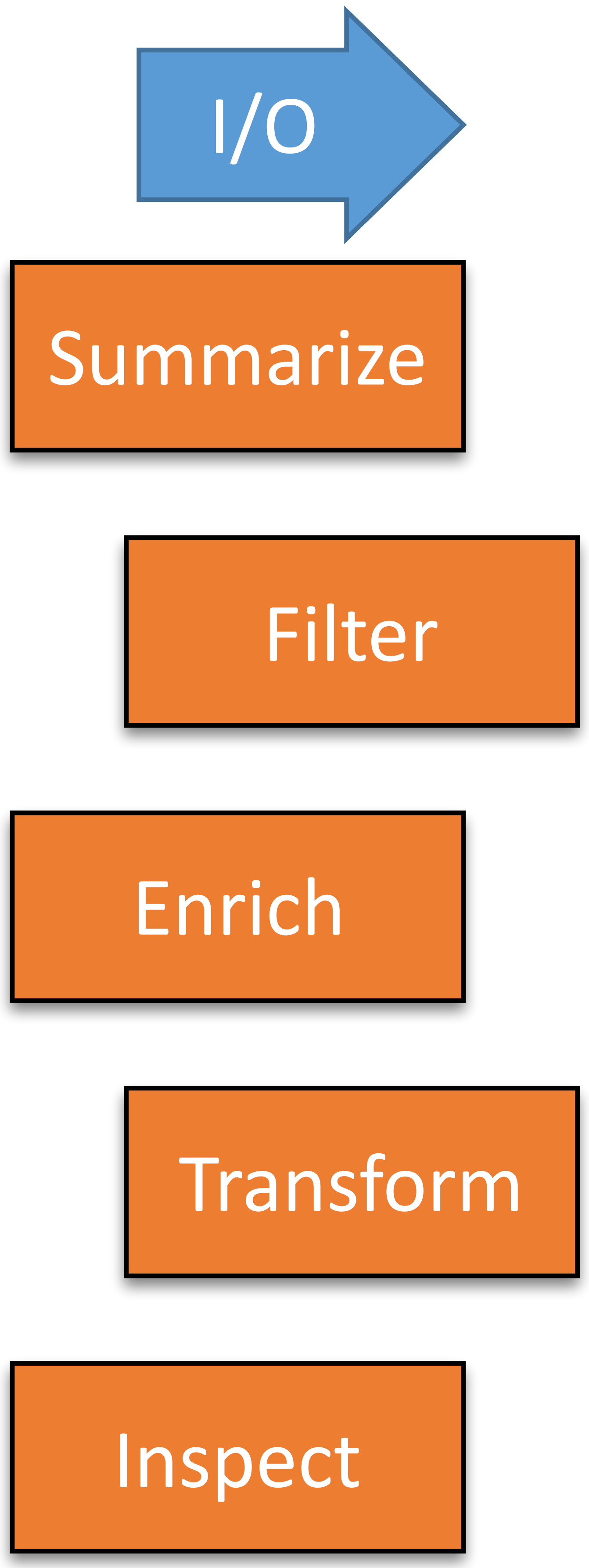


- Stay in memory
- Compute in small, reversible steps
- Enable introspection (code and state)
- Use other people's code
- Avoid boilerplate
- Maximize $\frac{t_{thinking}}{t_{thinking} + t_{coding} + t_{waiting}}$

So, basically Python…

…but fast

Hypothesis Testing on 50 Billion Records



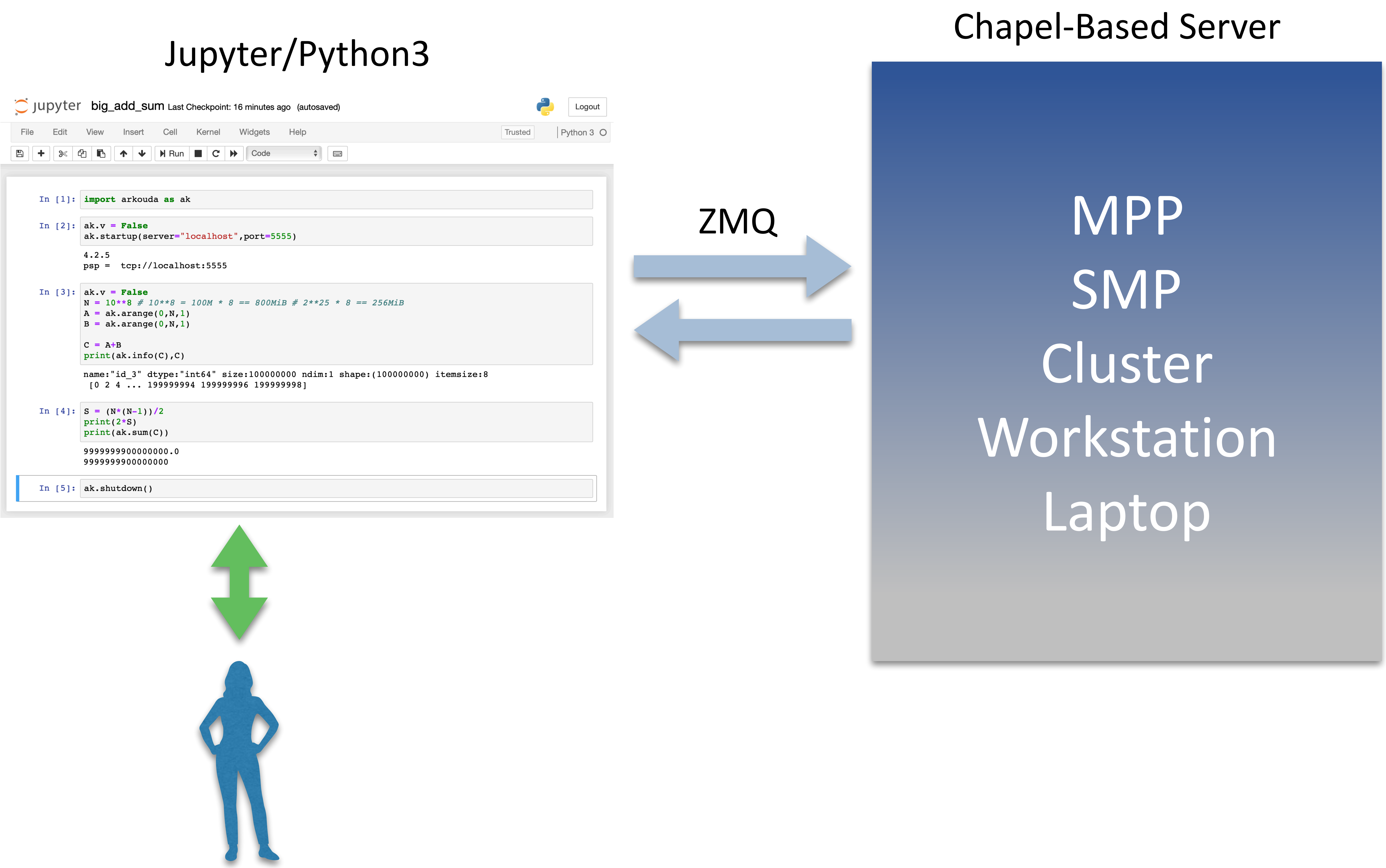
Operation	Example	Approximate Time (seconds)
Read from disk	A = ak.read_hdf()	30-60
Scalar Reduction	A.sum()	< 1
Histogram	ak.histogram(A)	< 1
Vector Ops	A + B, A == B, A & B	< 1
Logical Indexing	A[A == val]	1 - 10
Set Membership	ak.in1d(A, set)	1
Gather	B = Table[A]	30 - 300
Group by Key	G = ak.GroupBy(A)	60
Aggregate per Key	G.aggregate(B, ‘sum’)	15
Get Item	print(A[42])	< 1
Export to NumPy	A[:10**6].to_ndarray()	2

- A, B are 50 billion-element arrays
- Timings measured on real data
- Hardware: Cray XC40
  - 96 nodes
  - 3072 cores
  - 24 TB
  - Lustre filesystem

HPC Shell !?!

- Vision: Expose HPC libraries to Python via Arkouda
  - FFT, Tensor decomposition, Graph algorithms, Solvers
  - Anything you could link into a Chapel application (via C or LLVM)
- Need to standardize a distributed array interface
- Need an "HPC shell"

Arkouda Design



Arkouda Implementation

- Python3 client and Chapel server
- Client implementation in Python3
  - pdarray class
  - rely on Python to reduce complexity
  - integrate with and use NumPy
- Server Implementation in Chapel
  - restricted interpreter
  - symbol table — in memory object store
  - rely on Chapel for the things it does well

Where to get Arkouda?

- GitHub: arkouda
- PyPI: arkouda

- Open source under the MIT license.

Conclusion

Load Terabytes of data…

… into a familiar, interactive UI …

… where standard DS operations …

… execute within the human thought loop …

… and interoperate with optimized libraries.

It’s not crazy.

Backup slides

Why HPC Enabled EDA?

- We have data analyses which need to be done at a much larger scale… because sampling to run at smaller scale alters what can be seen in the data
- We need to enable our data scientists with tools they know… so why not co-opt an interface or two
- “Python is the new bash”
- Because we can and it’s fun!

Arkouda Startup

1) In terminal:

```
> arkouda_server -nl 96

server listening on hostname:port
```

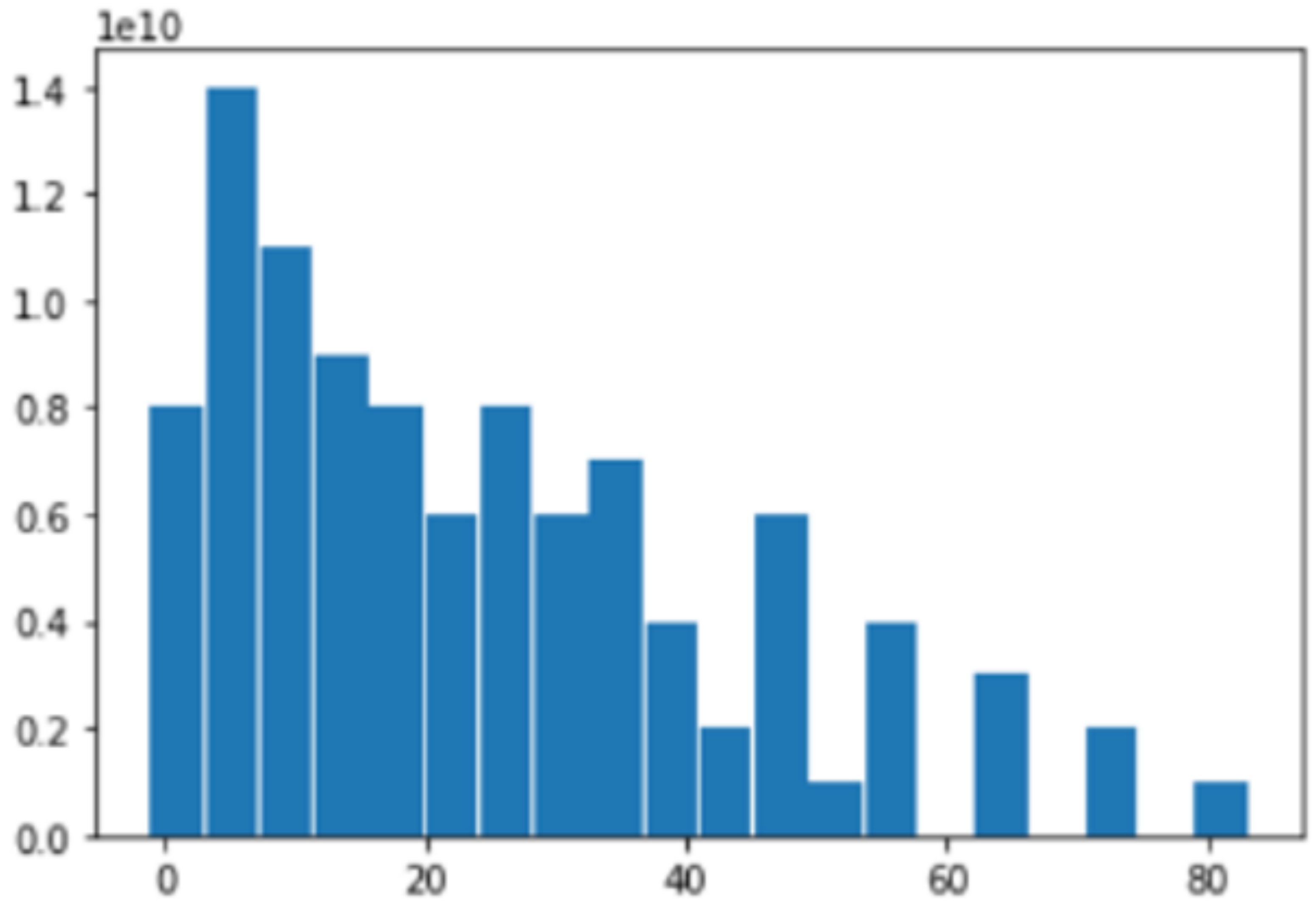
2) In Jupyter:

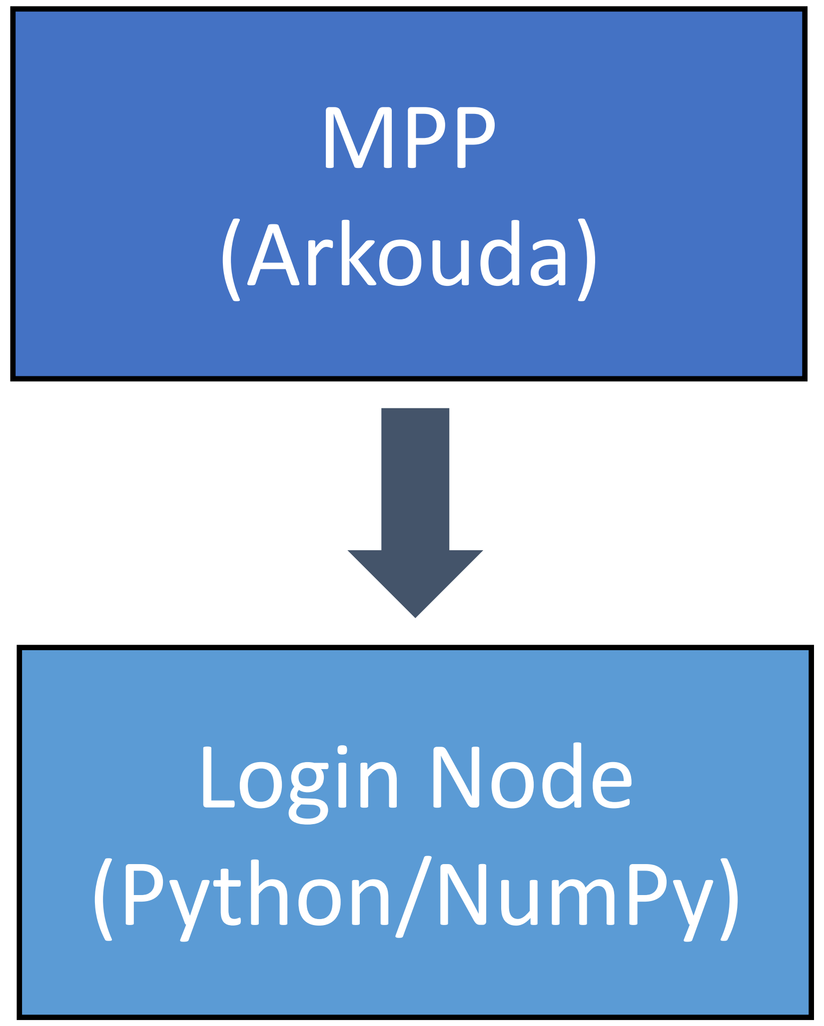
```
In [2]: import arkouda as ak
        ak.connect(hostname, port)

        4.2.5
        psp =  tcp://nid00104:5555
        connected to tcp://nid00104:5555
```

Data Exploration with Arkouda and NumPy

In [9]:
```
A = ak.randint(0, 10, 10**11)
B = ak.randint(0, 10, 10**11)
C = A * B
hist = ak.histogram(C, 20)
Cmax = C.max()
Cmin = C.min()
```
executed in 3.96s, finished 13:45:28 2019-09-12

In [10]:
```
bins = np.linspace(Cmin, Cmax, 20)
_ = plt.bar(bins, hist.to_ndarray(), width=(Cmax-Cmin)/20)
```
executed in 193ms, finished 13:45:28 2019-09-12





Slightly more complicated Arkouda example

RMAT Gen

```python
#!/usr/bin/env python3
import arkouda as ak

# generate rmat graph edge-list as two pdarrays
def gen_rmat_edges(lgNv, Ne_per_v, p, perm=False):
    # number of vertices
    Nv = 2**lgNv
    # number of edges
    Ne = Ne_per_v * Nv
    # probabilities
    a = p
    b = (1.0 - a)/ 3.0
    c = b
    d = b
    # init edge arrays
    ii = ak.ones(Ne,dtype=ak.int64)
    jj = ak.ones(Ne,dtype=ak.int64)
    # quantites to use in edge generation loop
    ab = a+b
    c_norm = c / (c + d)
    a_norm = a / (a + b)
    # generate edges
    for ib in range(1,lgNv):
        ii_bit = (ak.randint(0,1,Ne,dtype=ak.float64) > ab)
        jj_bit = (ak.randint(0,1,Ne,dtype=ak.float64) > (c_norm * ii_bit + a_norm * (~ ii_bit)))
        ii = ii + ((2**(ib-1)) * ii_bit)
        jj = jj + ((2**(ib-1)) * jj_bit)
    # sort all based on ii and jj using coargsort
    # all edges should be sorted based on both vertices of the edge
    iv = ak.coargsort((ii,jj))
    # permute into sorted order
    ii = ii[iv] # permute first vertex into sorted order
    jj = jj[iv] # permute second vertex into sorted order
    # to premute/rename vertices
    if perm:
        # generate permutation for new vertex numbers(names)
        ir = ak.argsort(ak.randint(0,1,Nv,dtype=ak.float64))
        # renumber(rename) vertices
        ii = ir[ii] # rename first vertex
        jj = ir[jj] # rename second vertex
    #
    # maybe: remove edges which are self-loops???
    #
    # return pair of pdarrays
    return (ii,jj)
```

-:--- connected_components.py Top (20,24) (Python)

BFS

```python
# src and dst pdarrays hold the edge list
# seeds pdarray with starting vertices/seeds
def bfs(src,dst,seeds,printLayers=False):
    # holds vertices in the current layer of the bfs
    Z = ak.unique(seeds)
    # holds the visited vertices
    V = ak.unique(Z) # holds vertices in Z to start with
    # frontiers
    F = [Z]
    while Z.size != 0:
        if printLayers:
            print("Z.size = ",Z.size," Z = ",Z)
        fZv = ak.in1d(src,Z) # find src vertex edges
        W = ak.unique(dst[fZv]) # compress out dst vertices to match and make them unique
        Z = ak.setdiff1d(W,V) # subtract out vertices already visited
        V = ak.union1d(V,Z) # union current frontier into vertices already visited
        F.append(Z)
    return (F,V)
```

Connected Components

```python
# src pdarray holding source vertices
# dst pdarray holding destination vertices
# printCComp flag to print the connected components as they are found
# edges needs to be symmetric/undirected
def conn_comp(src, dst, printCComp=False, printLayers=False):
    unvisited = ak.unique(src)
    if printCComp: print("unvisited size = ", unvisited.size, unvisited)
    components = []
    while unvisited.size > 0:
        # use lowest numbered vertex as representative vertex
        rep_vertex = unvisited[0]
        # bfs from rep_vertex
        layers,visited = bfs(src,dst,ak.array([rep_vertex]),printLayers)
        # add verticies in component to list of components
        components.append(visited)
        # subtract out visited from unvisited vertices
        unvisited = ak.setdiff1d(unvisited,visited)
        if printCComp: print("  visited size = ", visited.size, visited)
        if printCComp: print("unvisited size = ", unvisited.size, unvisited)
    return components

ak.connect(server="localhost", port=5555)
(ii,jj) = gen_rmat_edges(20, 2, 0.03, perm=True)
src = ak.concatenate((ii,jj))# make graph undirected/symmetric
dst = ak.concatenate((jj,ii))# graph needs to undirected for connected components to work
components = conn_comp(src, dst, printCComp=False, printLayers=False) # find components
print("number of components = ",len(components))
print("representative vertices = ",[c[0] for c in components])
ak.shutdown()
```

-:--- connected_components.py Bot (58,0) (Python)

Python Implementation Details

- Python pdarray class: a shim for the distributed array on the Arkouda server
  - Stores server-side name of array
  - Has a NumPy-like dtype
  - Has methods that translate operators into server commands
- Arkouda relies on Python to reduce complexity
  - Scoping
  - Reference counting
  - Garbage collection
  - Exceptions
- Arkouda integrates with and uses NumPy
  - Dtypes
  - Argument validation
  - Type conversion

Chapel Implementation Details

- A restricted Chapel interpreter:
  - Symbol table holding multi-type array wrappers
  - Code to parse commands from Python and select functions, operators, and types
- Chapel does some things really well
  - Makes parallelism easy (often implicit!)
  - Abstracts away inter-node communication and data layout
  - Compiler templates some functions
  - Allows dynamic casts from generic arrays to typed arrays
- But some things are hard
  - Large “select” statements for choosing functions, operators, types (an issue for all statically-typed languages)
  - Long compile times
- Far too many details to cover here…