Arkouda

Interactive Supercomputing for Data Science

Dr. William Reus

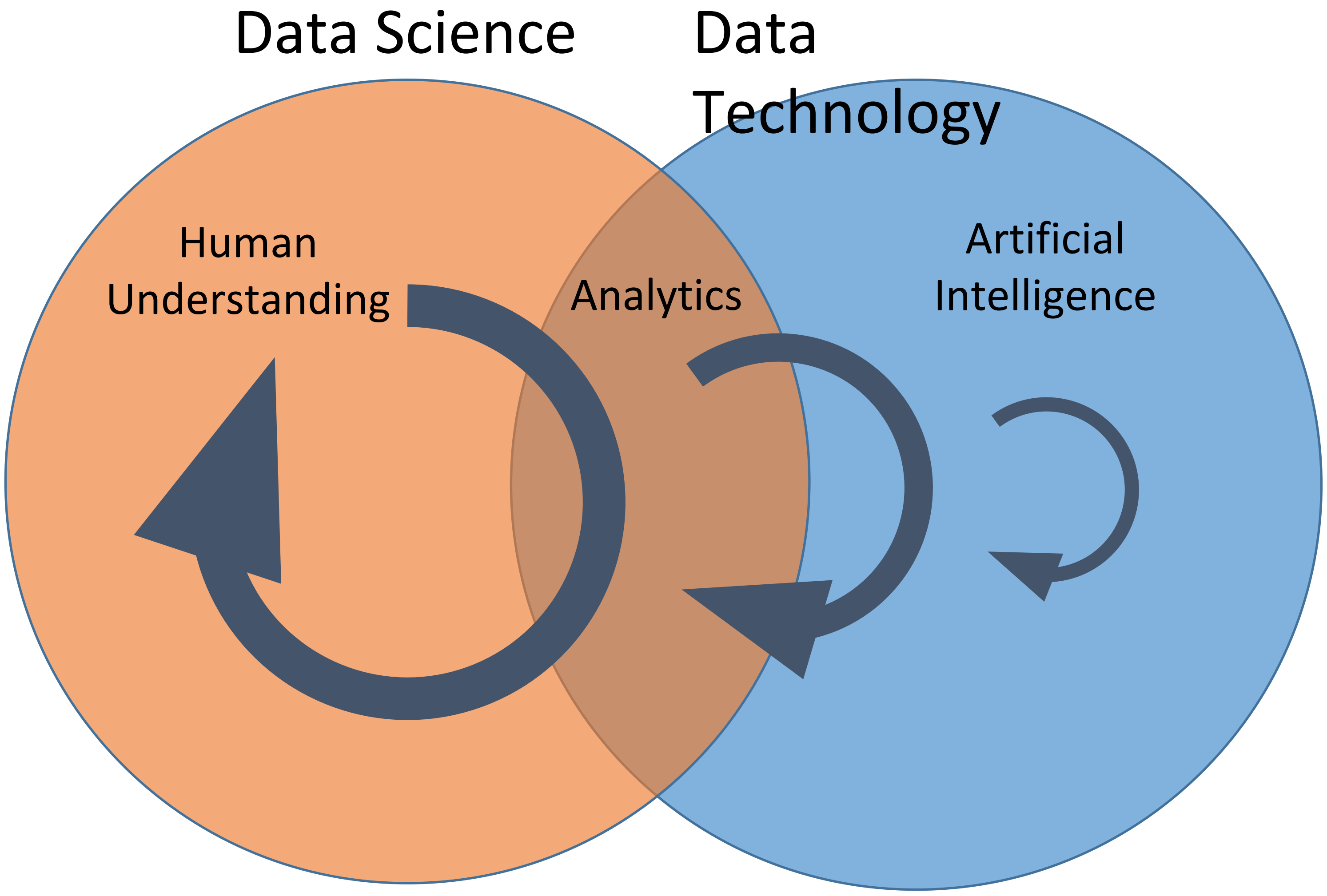
US Department of Defense

https://github.com/mhmerrill/arkouda

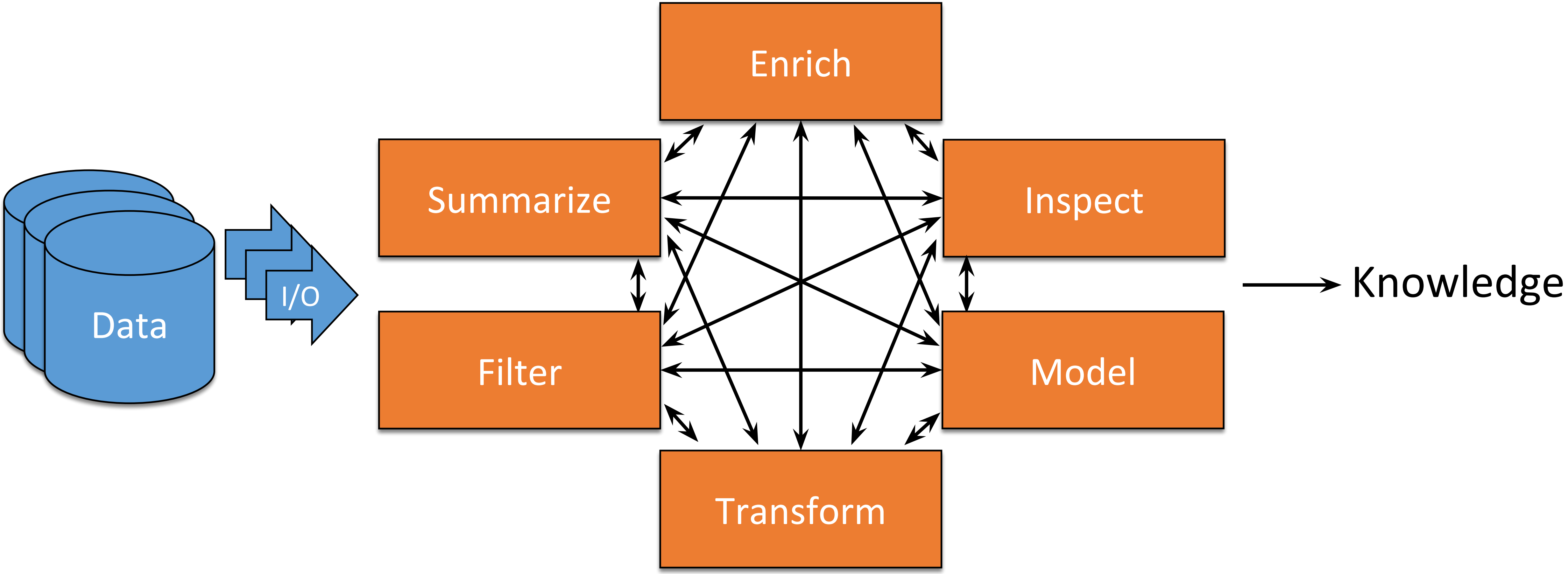
Data *Science*?

Data science proper is:

- Fundamental
- Difficult
- Computationally intensive
- Underemphasized

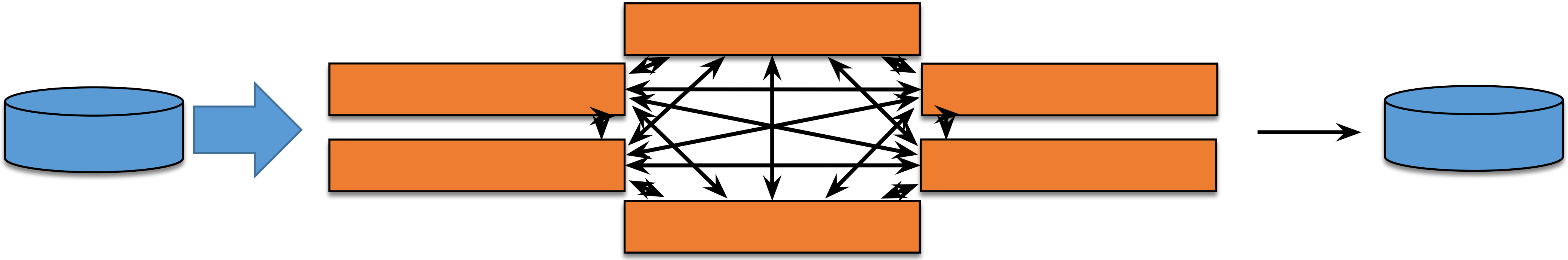


Understanding Physics of Datasets



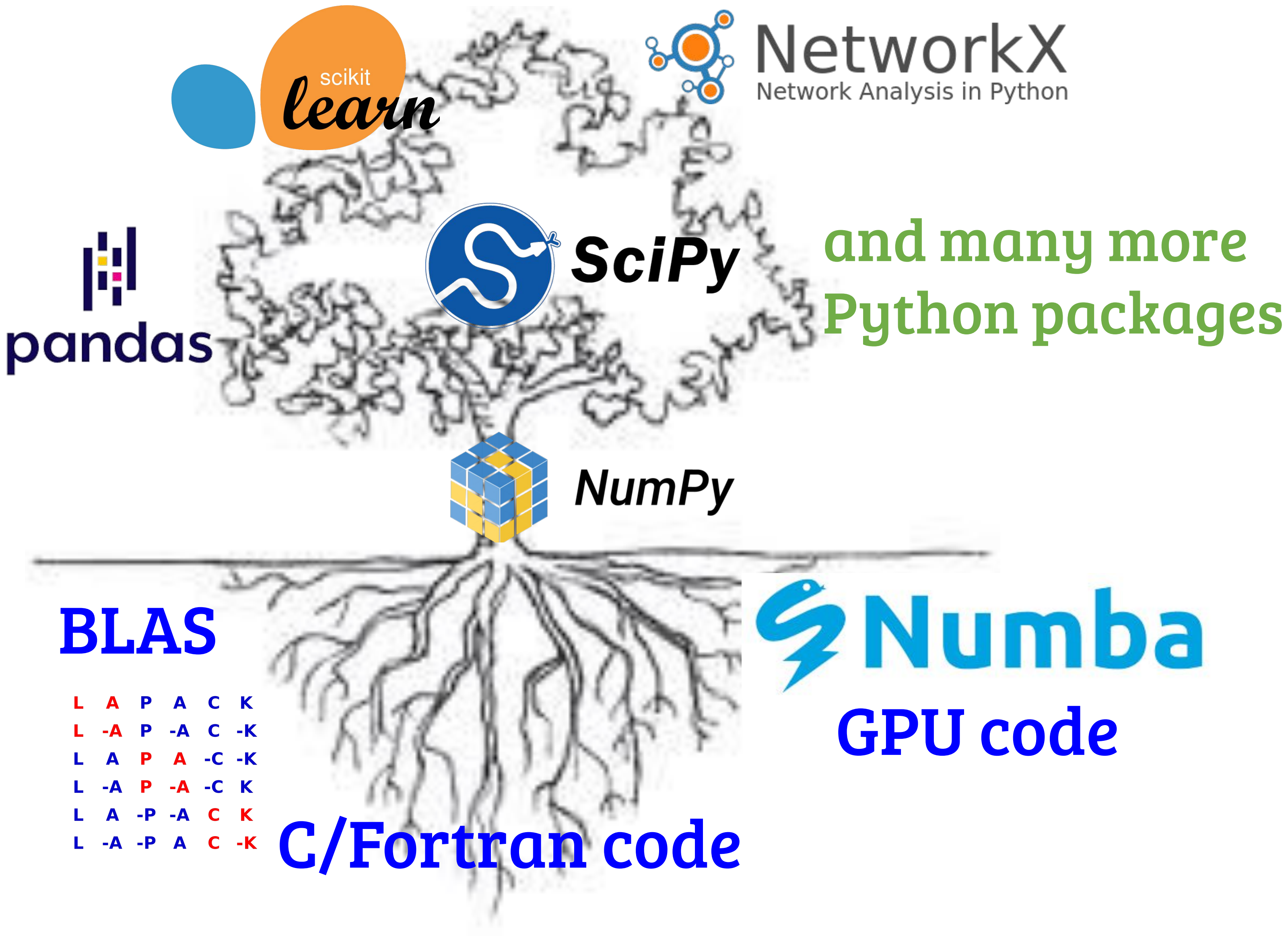
Many names: Exploratory Data Analysis, Data Wrangling, Data Modeling, etc.

Data Science Demands Interactivity

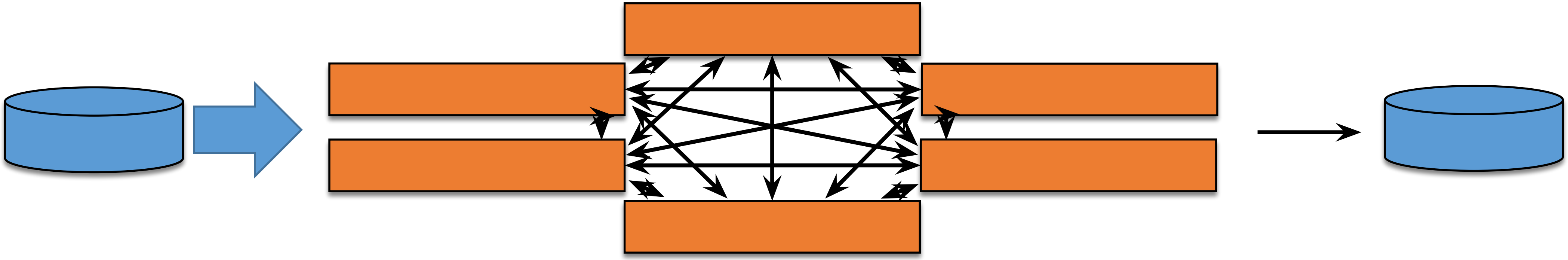


- Productivity with just enough performance
    - No compilation
    - No intermediate I/O
    - No writing boilerplate code
    - *Fast enough* to stay within thought loop
- Interactive Python on a large server satisfies these criteria for datasets up to 10-100 GB

Python Is Not Really Python

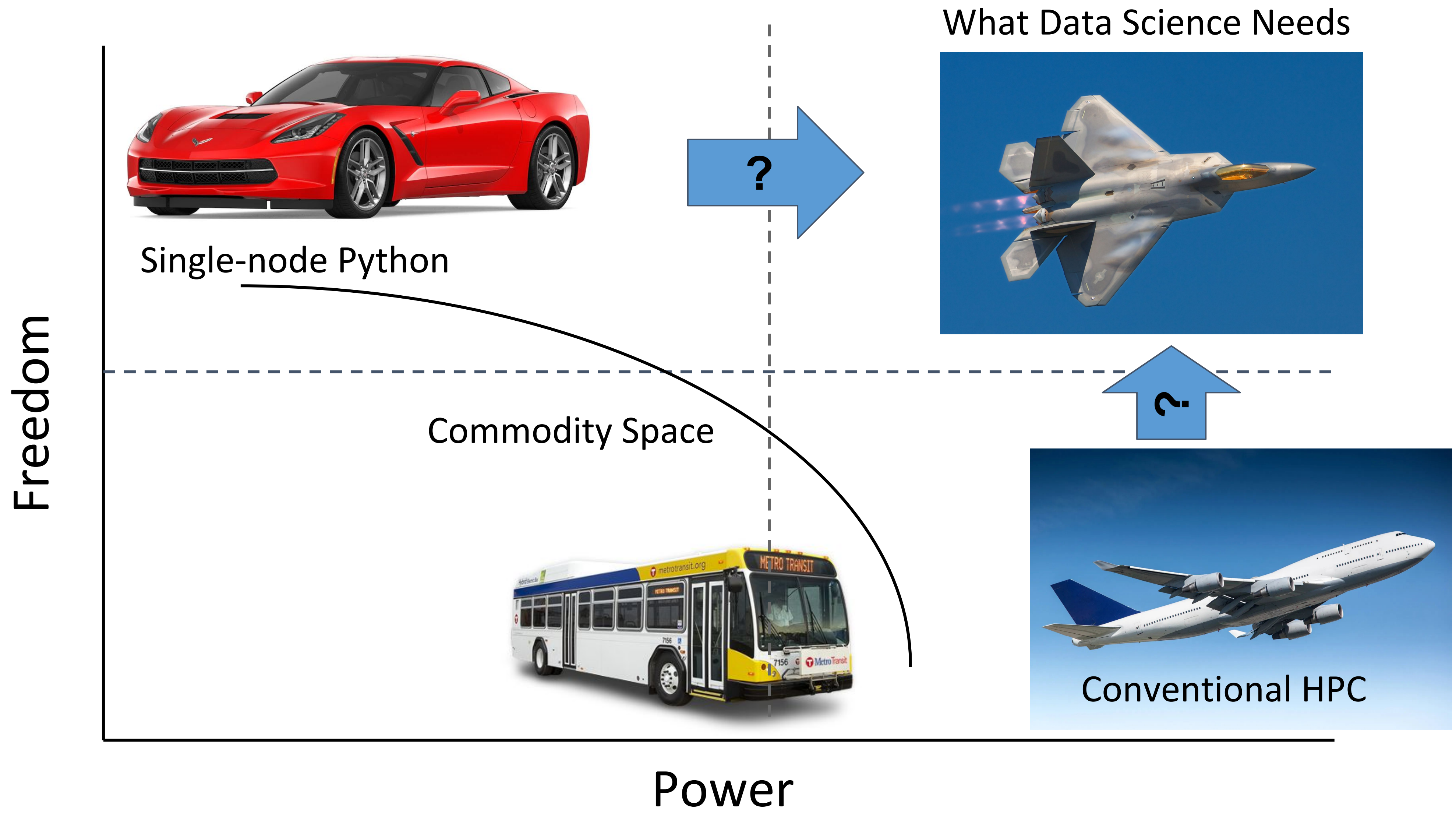


Data Science Demands Scaling



- Must use the whole dataset
  - Unbiased sampling of large datasets is difficult
  - Even unbiased sampling eliminates rare and high-order effects
  - Physics of most datasets are global, not local
- Datasets have outgrown (normal) computers
  - Server memory: ~ 1 TB
  - Many datasets > 10 TB

Dilemma: Interactivity vs. Scaling



Can We Fly an HPC?

Load Terabytes of data…

… into a familiar, interactive UI …

… where standard data science operations …

… execute within the human thought loop …

… and interoperate with optimized libraries.

Arkouda

Load Terabytes of data…

… into a familiar, interactive UI …

… where standard data science operations …

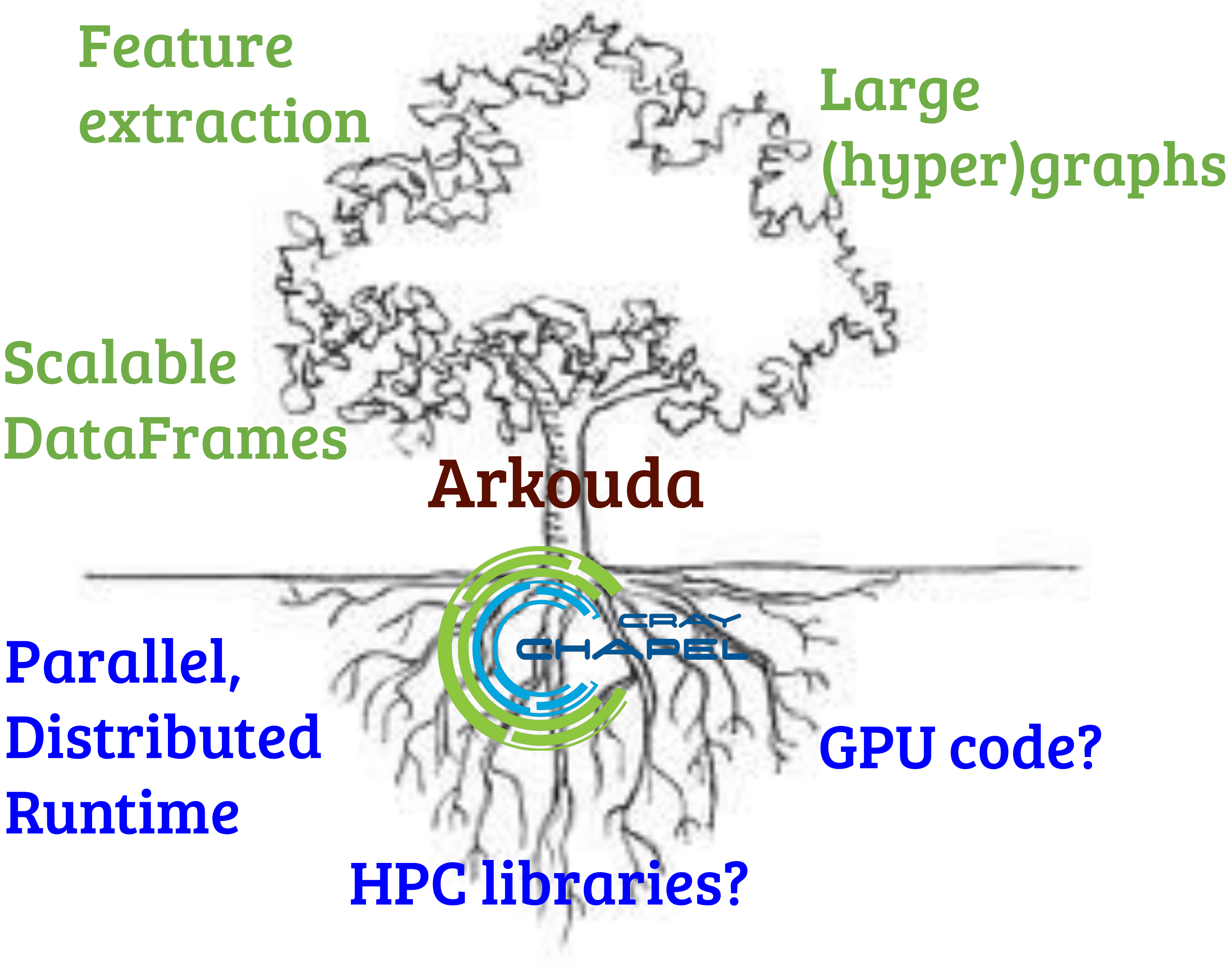
… execute within the human thought loop …

… and interoperate with optimized libraries.

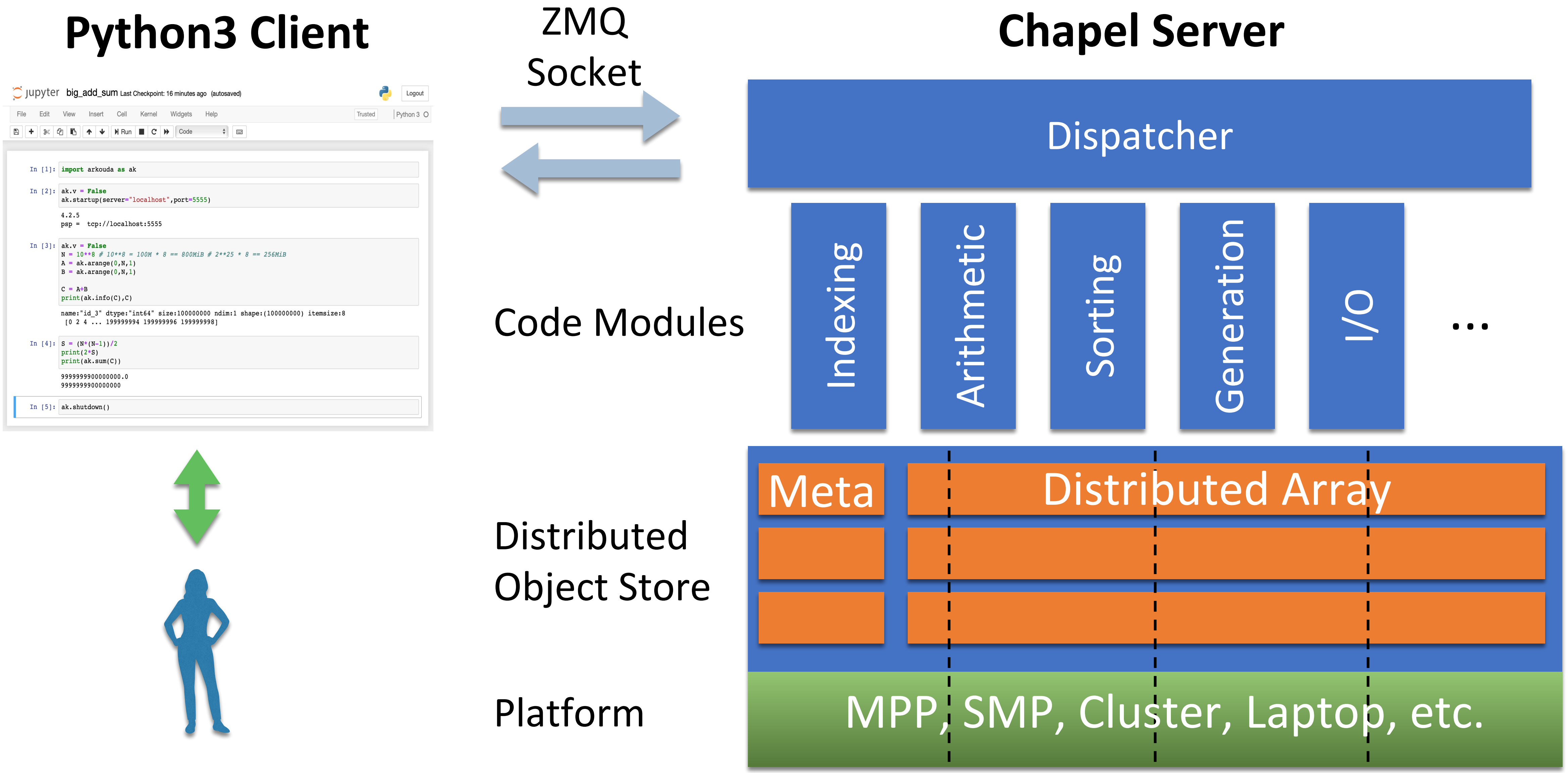
Arkouda: an HPC shell for data science

- Chapel backend (server)
- Jupyter/Python frontend (client)
- NumPy-like API

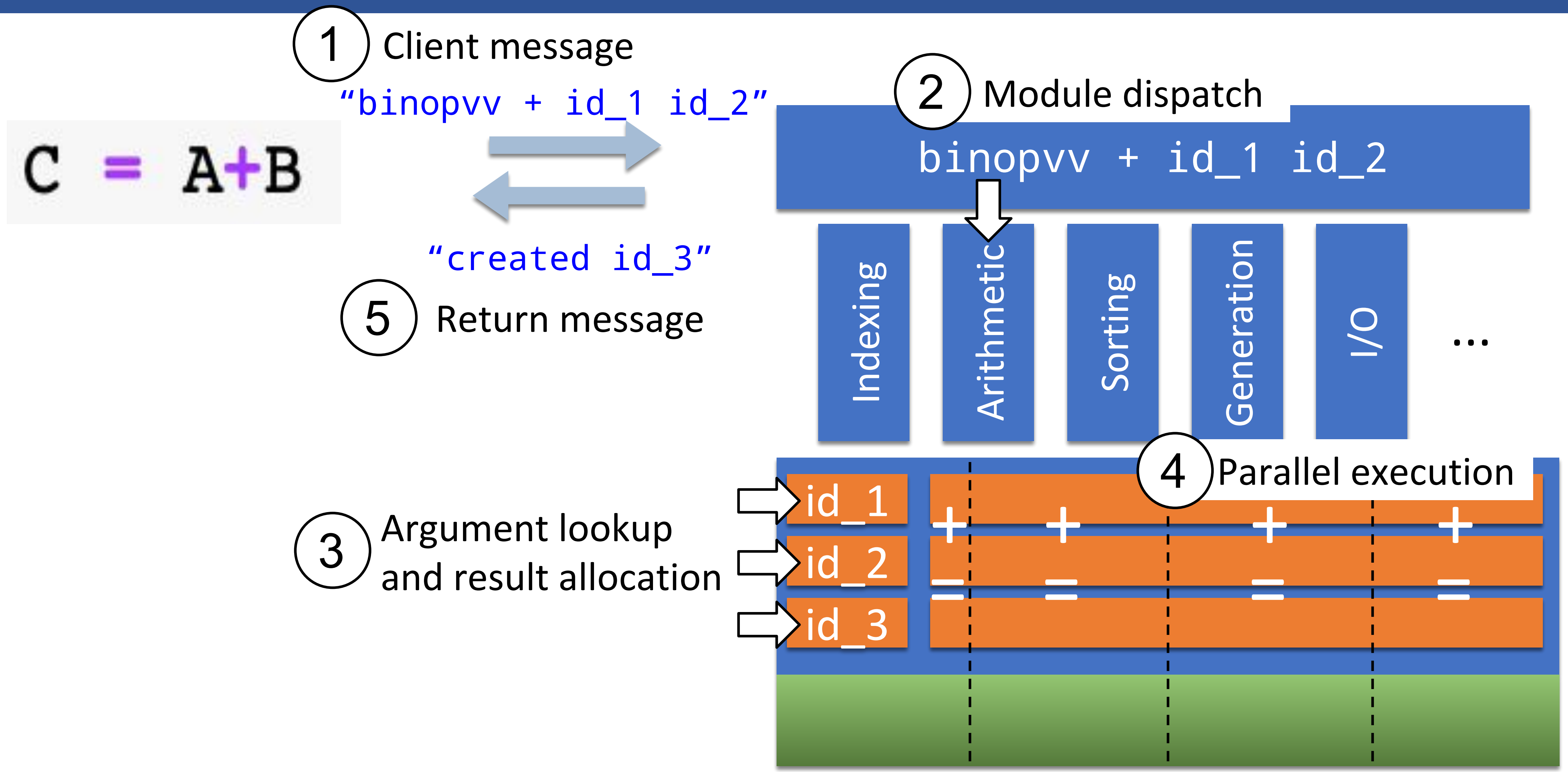
Arkouda: NumPy for HPC



Arkouda Design



A Chapel Interpreter



Client Handles Bookkeeping

```
C = A+B
```

- A, B, C are instances of `pdarray` class
  - attributes store metadata
    - size
    - data type (subset of NumPy dtypes)
    - server-side name
  - methods issue server commands
    - e.g. operator overloads
    - object deletion issues server command to free array data
- Client language (python) handles
  - scoping
  - garbage collection
  - reference counting
  - exceptions

Chapel Is Unique

- Productivity
  - Parallelism and locality are first-class citizens
  - Arkouda server = 12k lines of code
- Performance
  - Single-threaded comparable to NumPy (C/Fortran)
  - Parallel, distributed comparable to C/OpenMP/MPI
- Portability
  - Develop on laptop, run on supercomputer

Where Does Arkouda Fit In?

- Unique approach: start with performance, build towards interactivity
- Arkouda uses the HPC
  - Scales well to at least 512 nodes / 18k cores
  - Exploits features of high-speed interconnects
  - Leverages parallel filesystems
  - All without user fine-tuning
- Current drawbacks
  - Still adding major features (e.g. authentication)
  - Only one I/O format (HDF5)
  - GPU support only for client

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
```

Toy Workflow

In [9]:

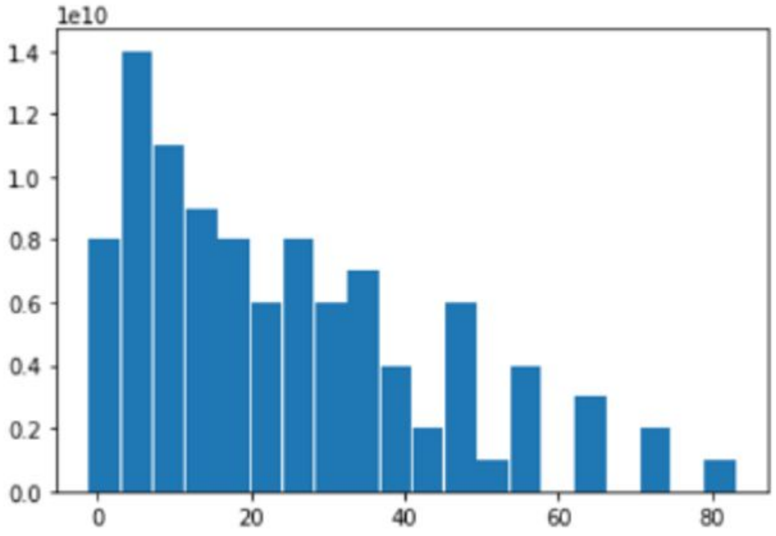
```
A = ak.randint(0, 10, 10**11)
B = ak.randint(0, 10, 10**11)
C = A * B
hist = ak.histogram(C, 20)
Cmax = C.max()
Cmin = C.min()
```

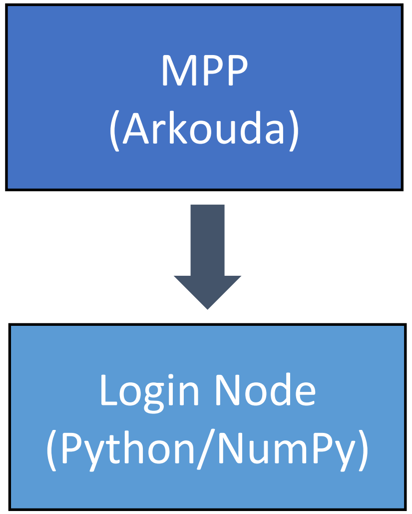
executed in 3.96s, finished 13:45:28 2019-09-12

In [10]:

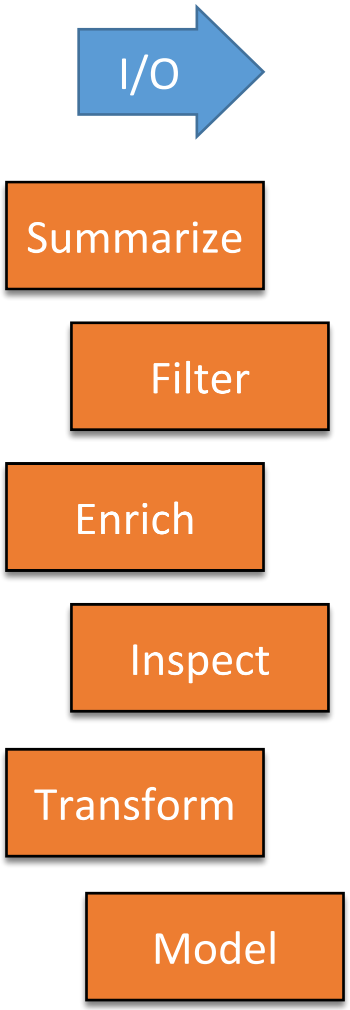
```
bins = np.linspace(Cmin, Cmax, 20)
_ = plt.bar(bins, hist.to_ndarray(), width=(Cmax-Cmin)/20)
```

executed in 193ms, finished 13:45:28 2019-09-12





Data Science on 50 Billion Records



Operation	Example	Approx. 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[B == val]	1 - 10
Set Membership	ak.in1d(A, set)	1
Gather	B = Table[A]	4 - 120
Get Item	print(A[42])	< 1
Sort Indices by Value	I = ak.argsort(A)	15
Group by Key	G = ak.GroupBy(A)	30
Aggregate per Key	G.aggregate(B, ‘sum’)	10

- A, B are 50 billion-element arrays of 32-bit values
- Timings measured on real data
- Hardware: Cray XC40
  - 96 nodes
  - 3072 cores
  - 24 TB
  - Lustre filesystem

Sorting is Critical

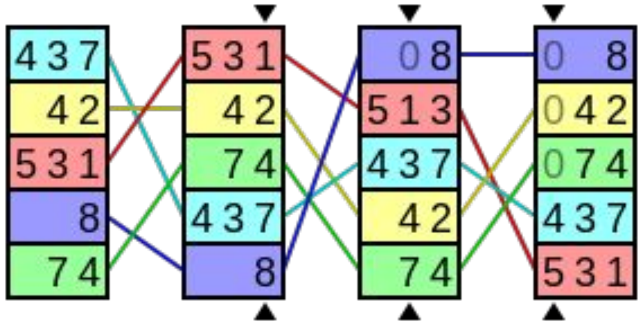
Sorting (`argsort` and `coargsort`) is the rate-limiting step in most arkouda workflows:

- Grouping tabular data by one or multiple columns
- Creating sparse matrices (graphs)
- Finding unique values and reindexing
- Extracting features for statistical testing
- Computing set operations

Sorting Is Critical

Arkouda uses a least-significant-digit radix sort

- Requires a fast interconnect
  - communication is $O(wn)$
- But runtime is independent of data distribution
  - best case = worst case = avg. case = $O(wn)$



https://www.growingwiththeweb.com/sorting/radix-sort-lsd/

$$w = \lceil \log_{radix} (\max - \min) \rceil$$

Sorting Scales

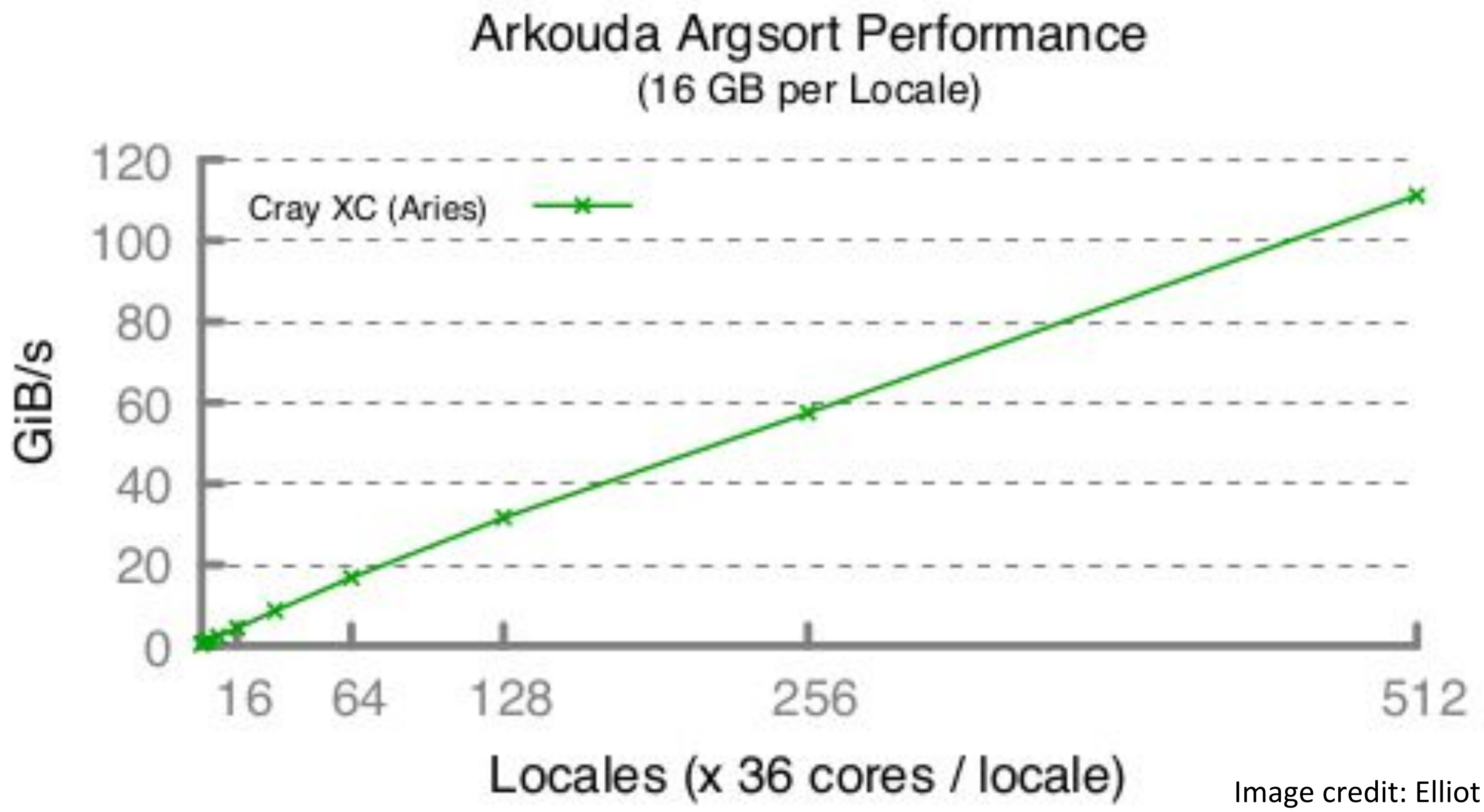


Image credit: Elliot Ronaghan, HPE

Performance Is Portable

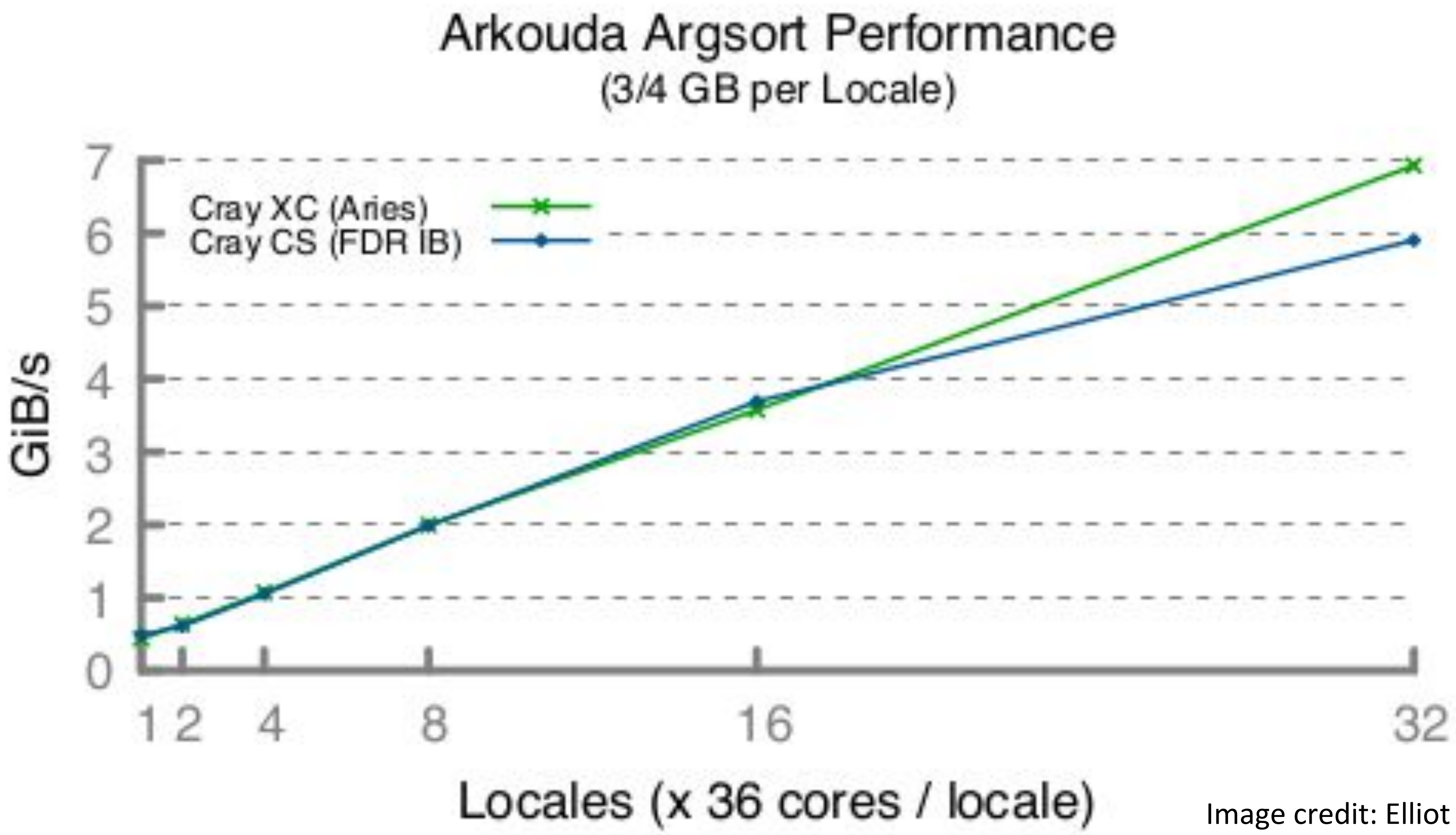
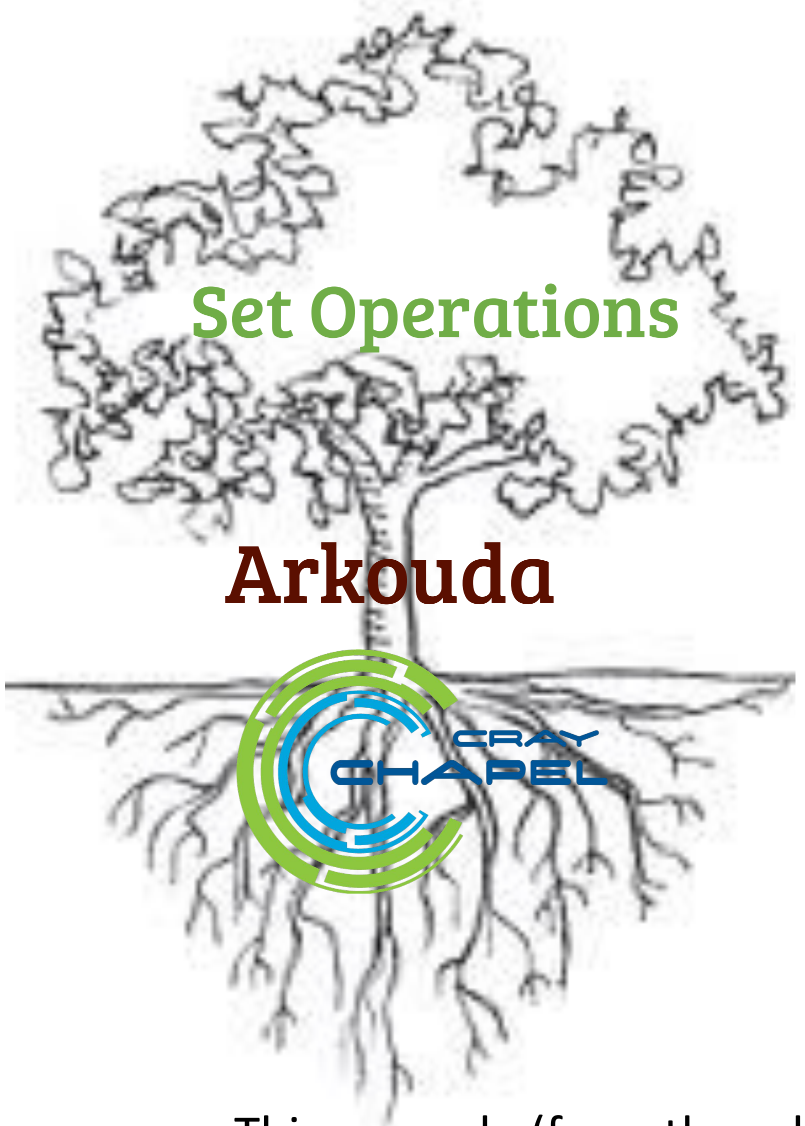


Image credit: Elliot Ronaghan, HPE

Climbing the Tree




```
>>> ak.intersect1d(ak.array([1, 3, 4]), ak.array([1, 2, 3]))
   array([1, 3])

def intersect1d(pda1, pda2, assume_unique=False):
   if isinstance(pda1, pdarray) and isinstance(pda2, pdarray):
       if pda1.size == 0:
           return pda1 # nothing in the intersection
       if pda2.size == 0:
           return pda2 # nothing in the intersection
       if not assume_unique:
           pda1 = unique(pda1)
           pda2 = unique(pda2)
       aux = concatenate((pda1, pda2))
       aux_sort_indices = argsort(aux)
       aux = aux[aux_sort_indices]
       mask = aux[1:] == aux[:-1]
       int1d = aux[:-1][mask]
       return int1d
   else:
       raise TypeError("must be pdarray {} or {}".format(pda1,pda2))
```

This example (from the arkouda source code) is very similar to `numpy.intersect1d`

Future Directions

- Leaves
  - Implement DataFrames
  - Add sparse linear algebra (GraphBLAS)
  - ???
- Trunk
  - Authentication
  - Data sharing and access control
  - Multi-user resource management?
- Roots
  - Link in FFT, tensor decomp., solvers, etc.
  - Need to standardize a distributed array interface with the HPC community

A New (Old) Perspective on HPC

Not Just This

But Also This

Acknowledgements

- Michael Merrill – inventor and lead developer
- Elliot Ronaghan – significant performance enhancements, scaling studies
- Chapel team – instrumental in helping arkouda use Chapel to the fullest
- All our contributors!

https://github.com/mhmerrill/arkouda