Arkouda Bulletin

A Year of Progress in Exploratory Data Analytics at Scale

Amanda Potts, Engin Kayraklioglu

ChapelCon '25
October 9, 2025

αρκούδα
massive scale
data science

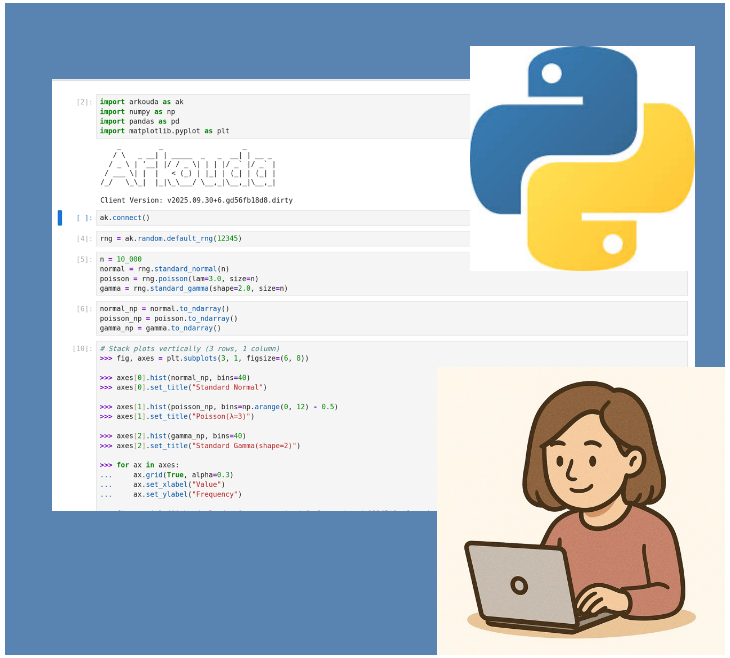
Outline

- Arkouda Overview & Introduction
- Success Stories
- Alignment to NumPy/Pandas
- Random Module
- Multi-Dimensional Data
- Parquet I/O Support
- Sparse Computations
- Outlook/Conclusion

Introduction

Arkouda: NumPy for Supercomputers

- **A Pythonic interface to high-performance computing**
  - Open-source framework for **exploratory data analysis at scale**
  - Combines **NumPy-like syntax** with **Chapel's distributed performance**
  - Operates **interactively from Python** — *no parallel programming required*
  - Handles **billions of elements** across **many nodes**, reproducibly



Arkouda: Where Python Meets Performance

αρκούδα
massive scale
data science

- **Bridging Productivity and Performance**
  - **Familiar** — mirrors NumPy, pandas, and SciPy semantics
  - **Scalability** — parallel computation over **massive arrays**
  - **Reproducible** — deterministic RNG and shuffle operations
  - **Extensible** — easy to add new Chapel functions via message framework
  - **Accessible** — installable via **Spack**; **Docker/Kubernetes** support under exploration
  - **Evolving** — ak.numpy, ak.pandas, ak.scipy
- **Bottom Line**
  - **Empowers** researchers to move seamlessly from prototyping in Python to analyzing terabytes interactively, without rewriting code.

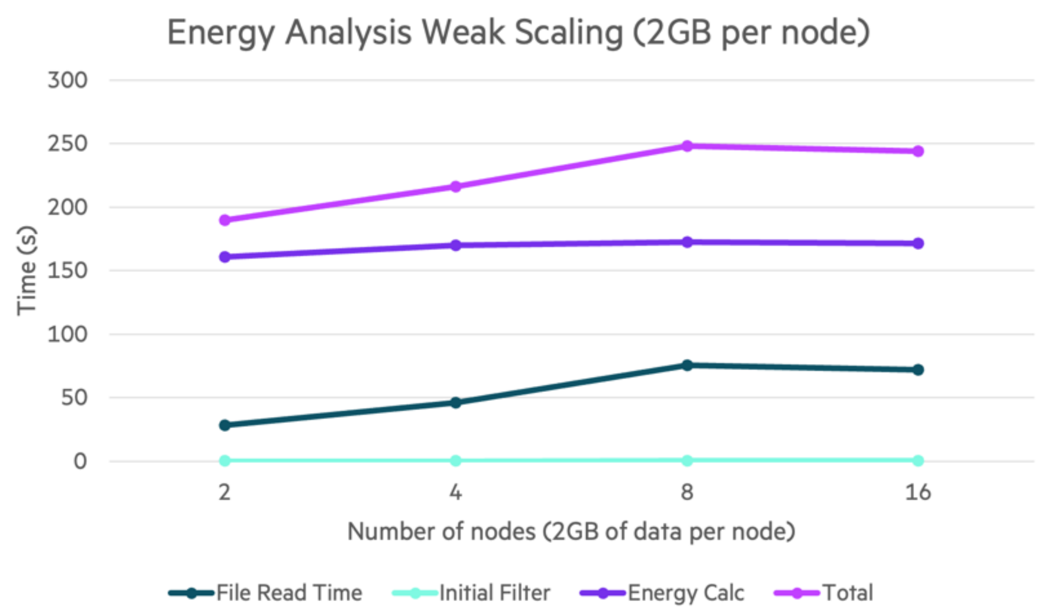
Success Stories

Telemetry Use Case 1

- How does energy-capping GPUs impact application performance?
- This work is a collaboration between our colleagues at HPE and ORNL
  - Using telemetry data from Frontier

Experiment details:

- A pandas script has been transliterated to Arkouda
  - Achieved 3.5x better performance on a single node
  - Same script also demonstrates good weak scaling
  - Note that pandas can't be used on multiple nodes at all

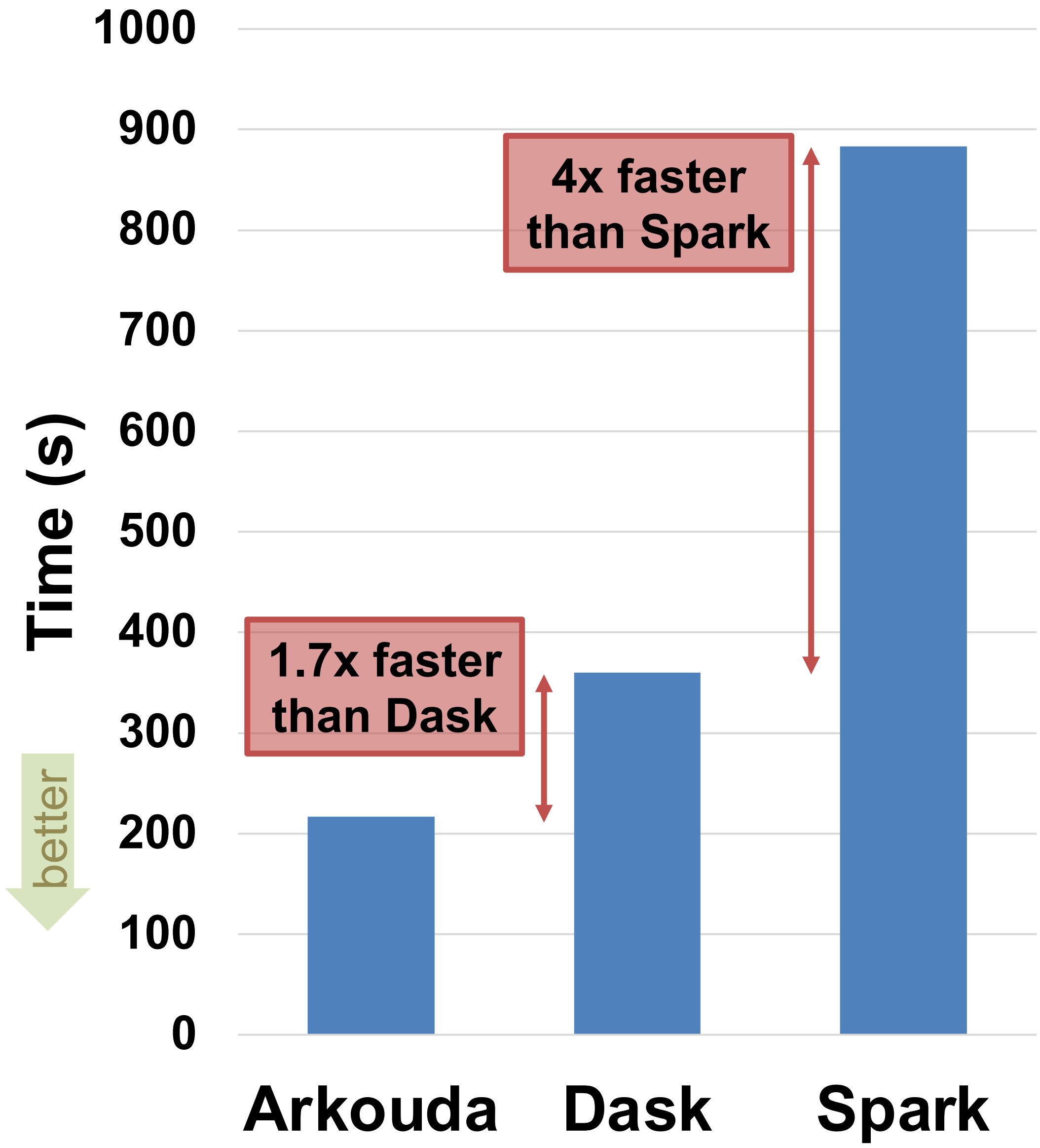


Telemetry Use Case 2

- What is the relationship between environment (e.g. temperature) and node failures?
- Imagine you have a very large server telemetry data, and information on failures, can you find any correlation?

Experiment details:

- 4TB of data stored in Parquet files
- Operations include:
    - Histogram
    - Mean, max
    - Covariance
- All experiments were run on 64 nodes of HPE Cray EX



Numpy & Pandas Alignment

Numpy Alignment

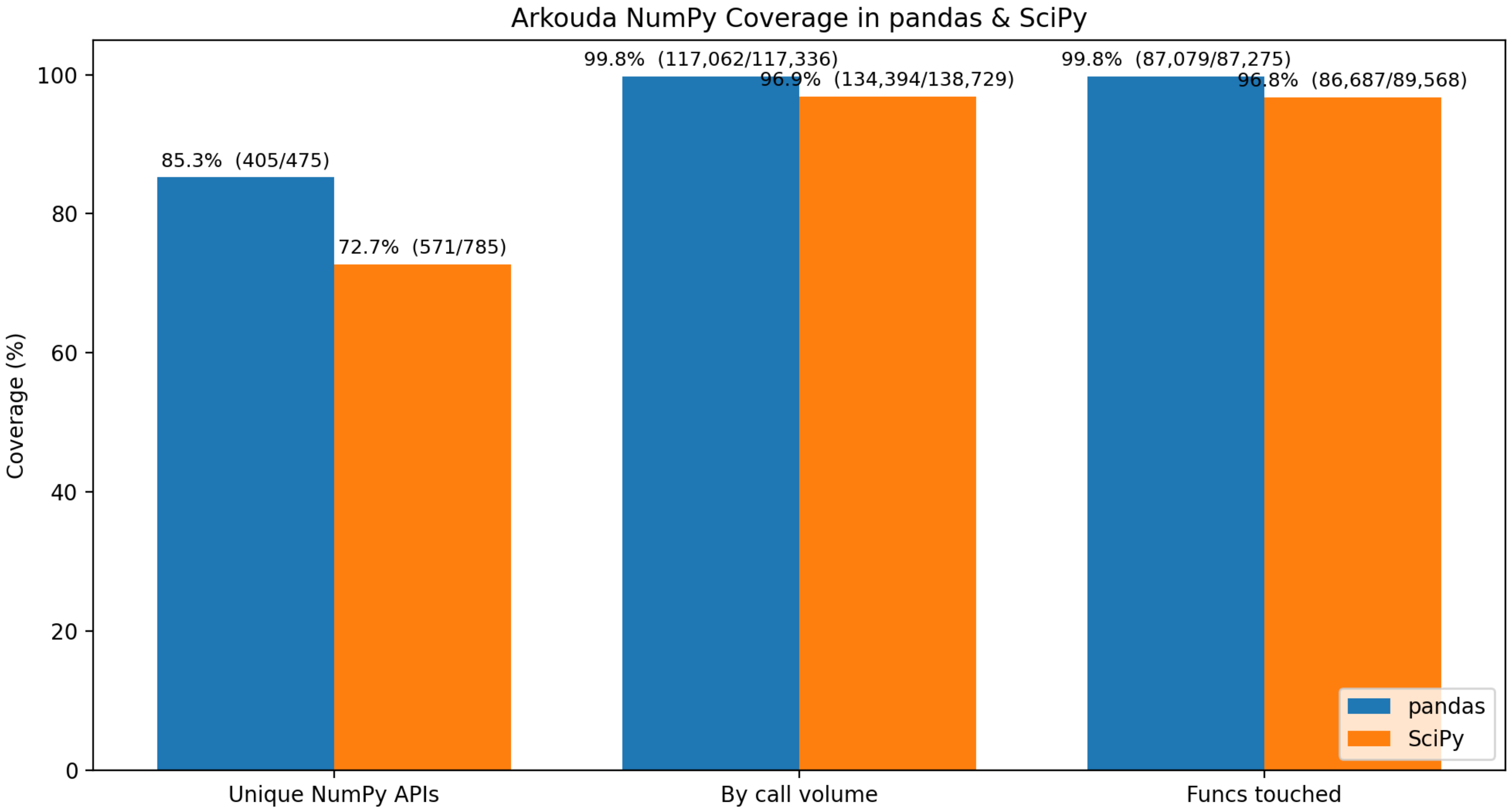
Strategy

- **Start with NumPy (foundation) →** pandas & SciPy depend on it; covering NumPy unlocks most downstream call paths.
- **Mirror NumPy APIs in ak.numpy → Users reuse NumPy muscle** memory (same names/args/semantics).
- **Use NumPy docs as the contract** → Solves the "web-search problem": answers from NumPy docs apply to Arkouda.
- **Reorganize into ak.numpy, ak.pandas, ak.scipy** → Clear place for each function; easier for contributors to navigate.
- **Rank by real usage (analyzed pandas/SciPy calls)** → High-impact first; all but 2 NumPy funcs used ≥10× in pandas functions are now supported.
- **Next: verify per-function parity** → Audit dtype promotion, broadcasting/axis, NA/Inf, and error behavior for exact NumPy match.


```
>>>
>>> import arkouda.numpy as np
>>>
>>> x = np.arange(6, dtype="uint64").reshape(2, 3)
>>> y = np.array([10, 20, 30], dtype="int64")
>>>
>>> (x + y).dtype    # promotion
dtype('float64')
>>>
>>> (x + y).shape    # broadcasting
(2, 3)
>>>
>>> np.sum(x + y, axis=0)
array([23.0 45.0 67.0])
>>>
```

Numpy Alignment

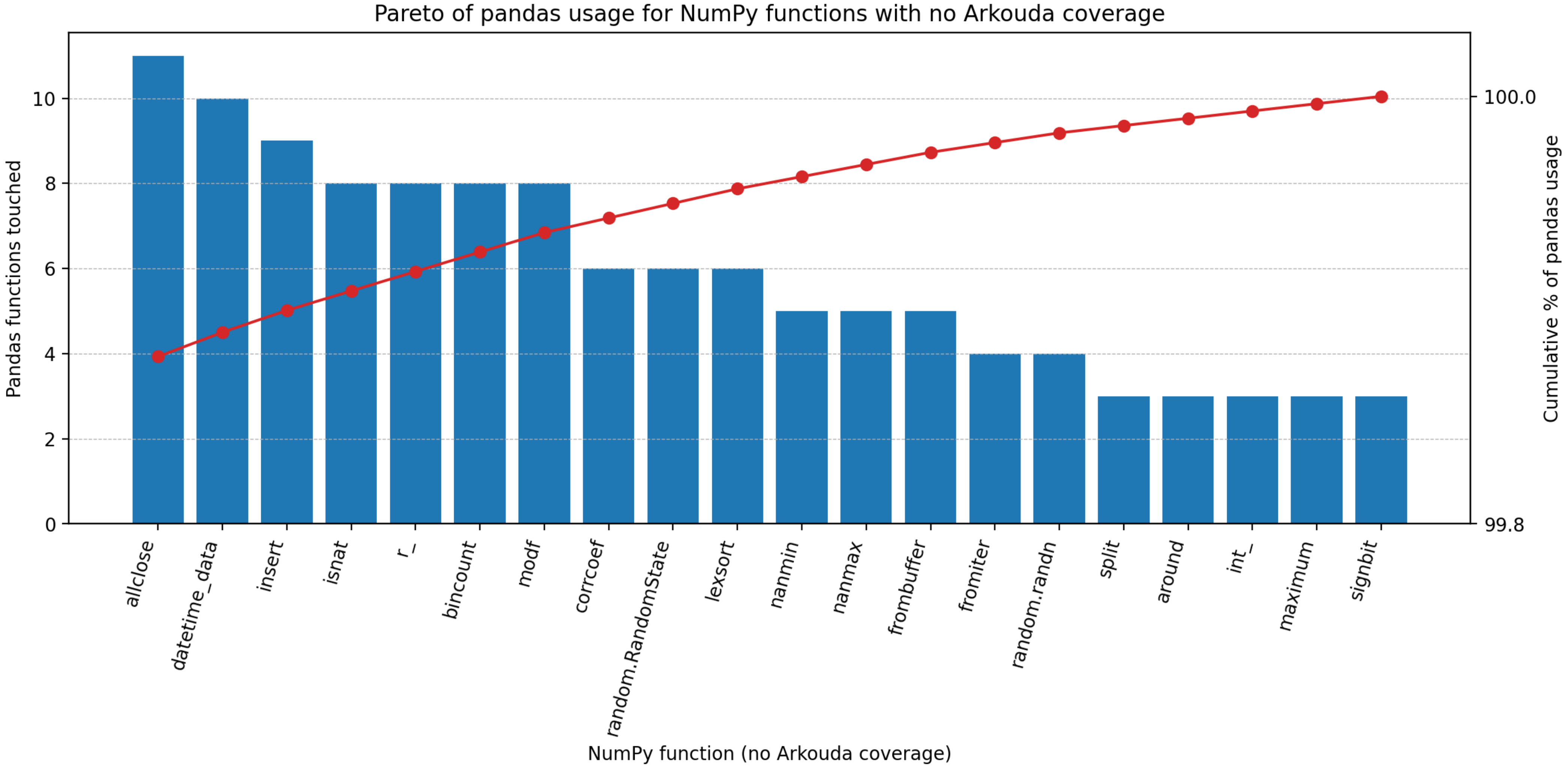
pandas & SciPy (what matters in practice)



Takeaway: Heavy hitters are covered; remaining work is a low-volume tail. Gaps by calls → pandas: 0.2%, SciPy: 3.1%.

Numpy Alignment

pandas & SciPy (what matters in practice)



Pandas ExtensionArray API (Arkouda)

Experimental

- **What It Is**
  - Arkouda-backed ExtensionArrays so pandas Series/DataFrame columns stay remote (numeric, bool, string, categorical).
  - Supports zero-copy construction where possible.
- **Why**
  - Enable scalable pandas workflows.
  - Avoid rewriting all of pandas.
- **What works today**
  - Column creation, indexing, equality, argsort, and common reductions.
  - Clean fallback to NumPy dtypes when needed.
- **Caveats**
  - Experimental — some pandas paths still call .to_numpy().
  - NA semantics and a few reductions incomplete in certain types.

```
>>>
>>> register_extension_dtype(ArkoudaInt64Dtype)
>>> register_extension_dtype(ArkoudaFloat64Dtype)
>>> register_extension_dtype(ArkoudaBoolDtype)
>>>
>>> x = pd.array([1, 2, 3], dtype="int64")
>>> y = pd.array([True, False, True], dtype="bool")
>>> z = pd.array([11, 22, 33], dtype="float64")
>>> x
ArkoudaArray([1 2 3])
>>>
>>> df1 = pd.DataFrame({"x":x, "y": y})
>>> df2 = pd.DataFrame({"x":x, "z": z})
>>>
>>> df3 = df1.merge(df2, on=["x"])
>>> df3
   x      y     z
0  1   True  11.0
1  2  False  22.0
2  3   True  33.0
>>>
>>> type(df["x"].values)
ArkoudaArray
>>>
```

Pandas ExtensionArray API (Arkouda)

- **Pandas ↔️ Arkouda Bridge (Joins/Merges/Groupby)**

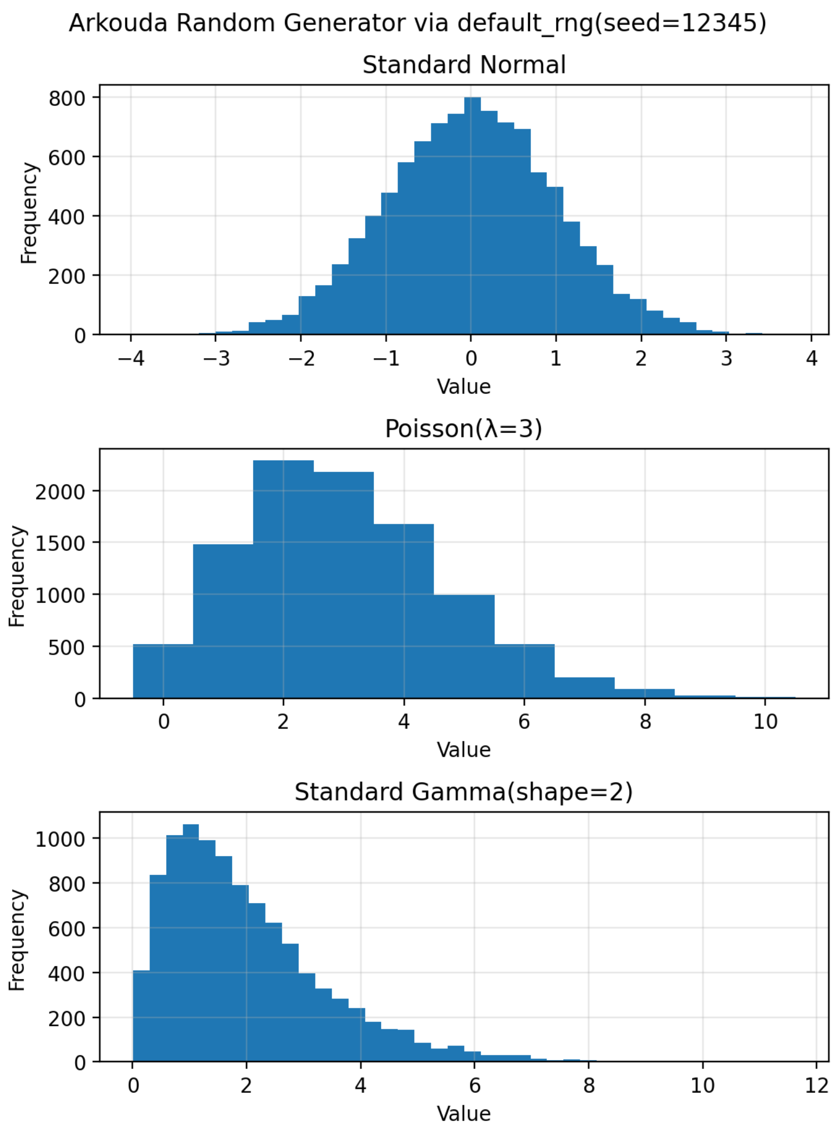
  - **Key point**
    - Arkouda already has **distributed** merges/groupby — need a **pandas bridge** that dispatches to Arkouda (avoid .to_numpy())
  - **Preferred: pandas accessor**
    - .ak on Series/DataFrame:
    - df.ak.merge(...), df.ak.groupby(...).agg(...)
  - **Other integration options**
    - **Subclassing** (pd.DataFrame/Series + mixin): natural syntax (df.merge) — but brittle vs pandas internals & upgrades
    - **Monkeypatching** (override DataFrame.merge, GroupBy.agg): fastest demo path — but risky, version-fragile; keep opt-in
  - **Status**
    - **Early experimental** EAs/dtypes exist; bridge layer TBD.

```
>>>
>>> df3 = df1.merge(df2, on=["x"])
>>> df3
   x      y     z
0  1   True  11.0
1  2  False  22.0
2  3   True  33.0
>>>
>>> df4 = df1.ak.merge(df2, on=["x"])
>>> df4
   x      y     z
0  1   True  11.0
1  2  False  22.0
2  3   True  33.0
>>>
```

Random Module

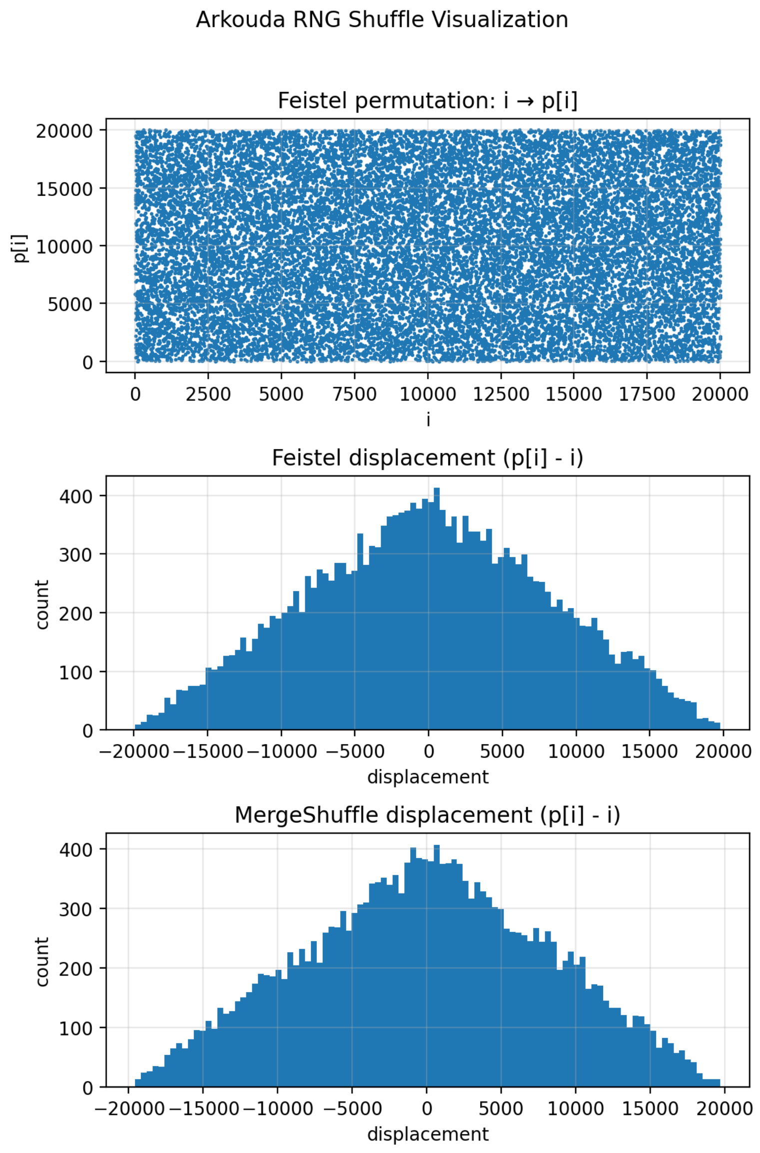
Random Module
What's New

- **ak.random.Generator** via default_rng(seed)
  - PCG64 backend, independent streams per dtype
- **Supported distributions**
  - integers, uniform, normal / lognormal
  - exponential / standard_exponential, poisson, standard_gamma
  - choice, permutation, shuffle
- **Method variants**
  - e.g. standard_normal(method={"zig","box"})
  - standard_exponential(method={"zig","inv"})
- **Legacy API (backward-compatible)**
  - ak.random.* functions (ak.rand, ak.randint, ak.uniform)
  - Now thin wrappers over the new Generator



Random Module
Reproducibility

- **Seeded & deterministic**
  - Stable results if locale count is unchanged (most ops)
- **Locale sensitivity**
  - Changing locale count → different draws/permutations
- **Locale-invariant exception**
  - shuffle(method="Feistel"): keyed permutation over [0, N)
- **Shuffle methods**
  - Fisher–Yates: simple, single-locale (testing / small data)
  - MergeShuffle: scalable, fully distributed; reproducible only if locale count fixed
  - Feistel: distributed, keyed, reproducible (not cryptographic)
- **Looking ahead**
  - Exploring stateless RNGs (Philox, Threefry) for locale-independent draws and per-element determinism



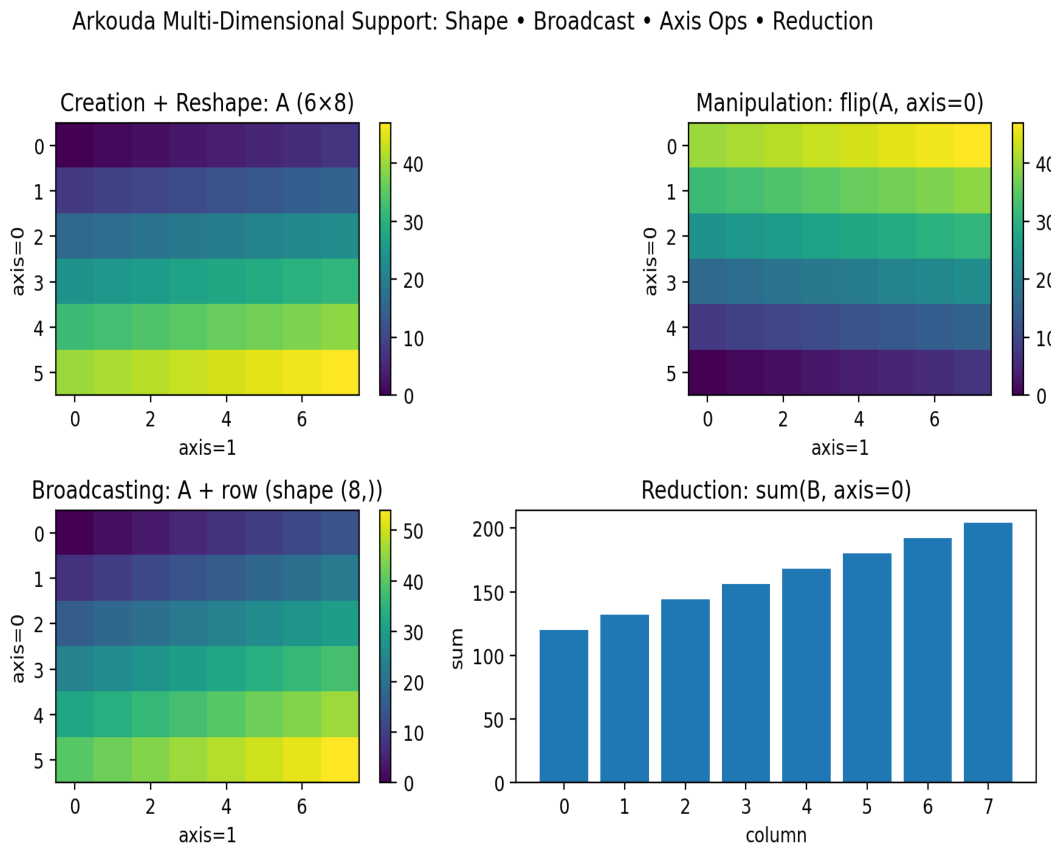
Multi-dim Support

Multi-Dimensional Arrays

What’s New

Multi-Dimensional Support

- **Creation**: array/zeros/ones/full/_like accept tuple shapes
- **Shape Ops**: reshape/flatten/squeeze with negative-axis support
- **Broadcasting**: rules aligned; centralized axis validation
- **Elementwise**: abs/cos/clz/isinf
- **Manipulation**: repeat, tile, flip are axis-aware for N-D; concatenate, where
- **Reduction**: sum/prod/min/max/cumsum/cumprod/diff
- **Linear Algebra**: matmul/dot/vecdot (mixed-rank matmuls not supported)
- **Sorting**: argsort/coargsort/sort support axis on numeric arrays




```
>>> A = ak.arange(6*8).reshape(6, 8)
>>> row = ak.arange(8)
```

Multi-Dimensional Arrays

Current Gaps/Next Up

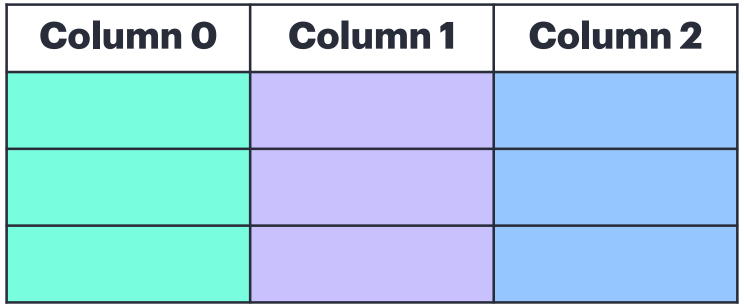
Feature Area	1-D	N-D	Notes/Status
Numeric Arrays	Yes	Yes	Core Ops Complete
Strings/Categorical	Yes	No	Not Implemented
Set Operations (intersect1d, union1d, etc...)	Yes	No	Currently 1-D only
Reductions (min/max etc...)	Yes	Partial	mink, argmink pending
Other	Yes	Partial	median, count_nonzero pending
Pandas Integration	Yes	No	No DataFrame/Series support

Parquet I/O Support

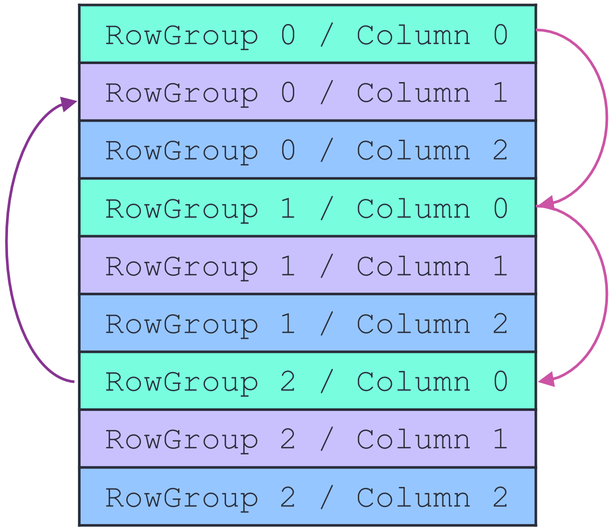
Parquet I/O Support

Previous All-Column Read Implementation

Logical Table:



Simplified Representation of Parquet File:

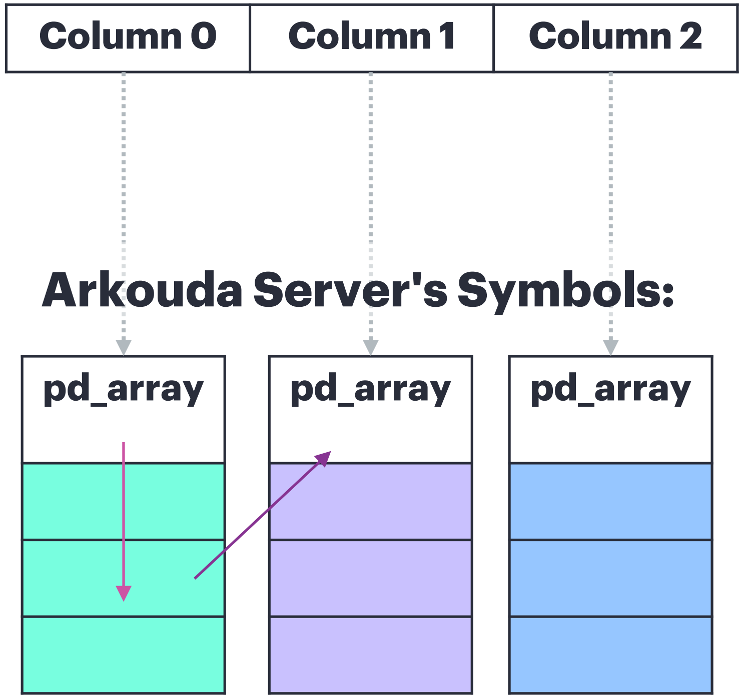


Previous implementation

Read a column

Jump back to read the next

Arkouda Client's Dataframe:



Parquet I/O Support

New All-Column Read Implementation

Logical Table:

Column 0	Column 1	Column 2

Simplified Representation of Parquet File:

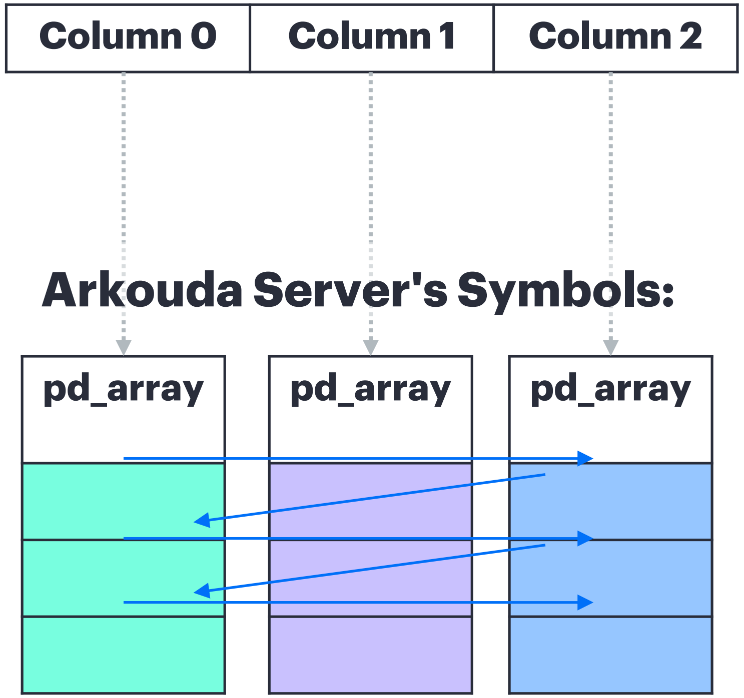
```
RowGroup 0 / Column 0
RowGroup 0 / Column 1
RowGroup 0 / Column 2
RowGroup 1 / Column 0
RowGroup 1 / Column 1
RowGroup 1 / Column 2
RowGroup 2 / Column 0
RowGroup 2 / Column 1
RowGroup 2 / Column 2
```

Current implementation

Read the file linearly

Populate columns at the same time

Arkouda Client's Dataframe:



Parquet I/O Support

Performance Results in Synthetic Benchmarks

Significantly Improved Read Performance, Especially with Multiple Columns

~400GBs of data, 5 columns, split into 128 files, read by 16 locales:

before (s)	after (s)	speedup (x)
16.69	9.38	**1.78**

Noticeable improvement with smaller numbers of columns

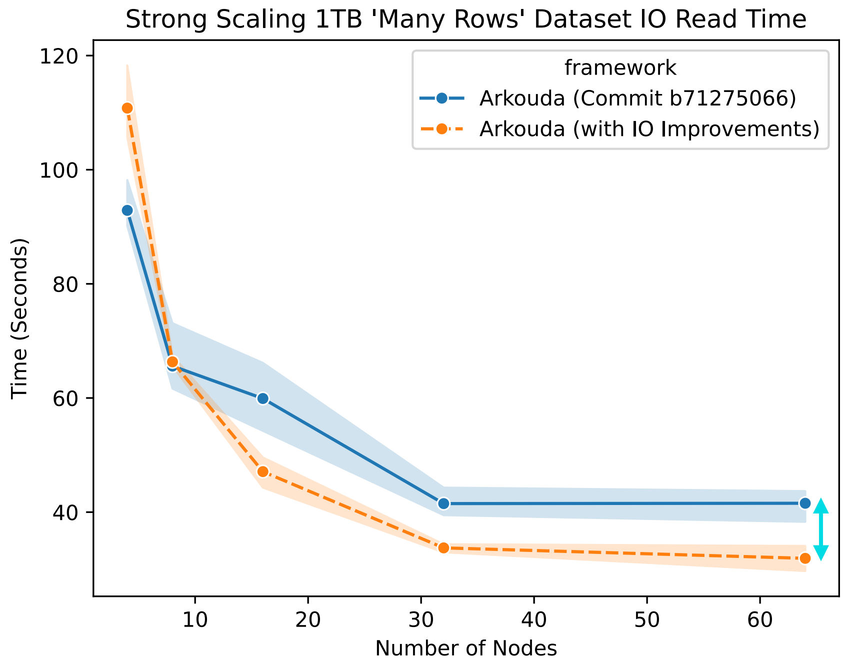
~30GBs of data, 1000 columns, split into 128 files, read by 16 locales:

before (s)	after (x)	speedup (x)
335.97	9.98	**33.67**

Gets more significant as number of columns increase

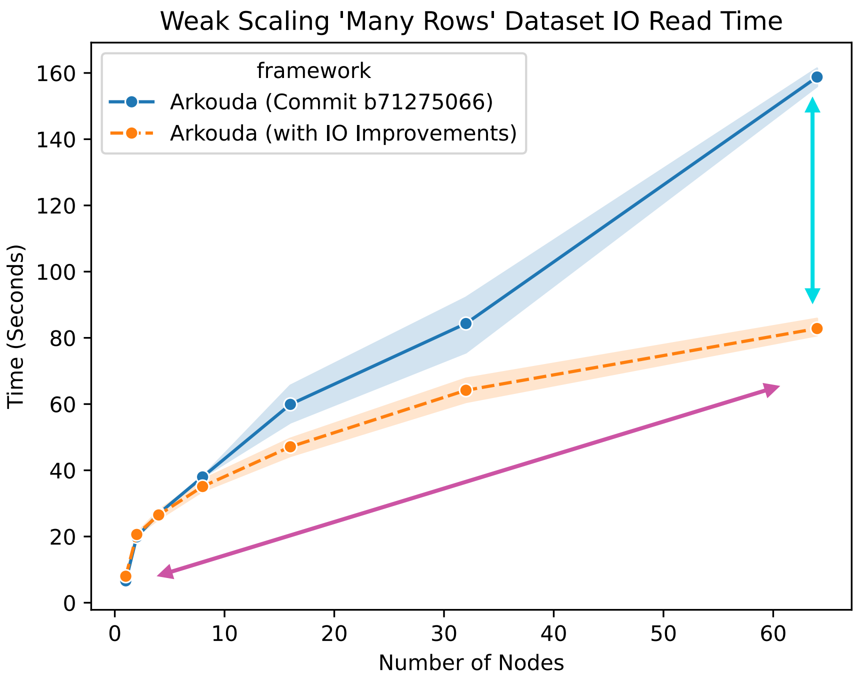
Parquet I/O Support

Performance Results from the Telemetry Use Case 2



More than 2x improvement at scale

Much better weak scaling behavior



Sparse Computations

Improving Arkouda's Sparse Linear Algebra Capabilities

Challenge: Can you create a distributed sparse domain & array pair,

- using 3 Arkouda pdarrays for rows, columns, and values,
- where the arrays are not necessarily sorted, nor contain unique data ?

Potential Answer: Well, of course! You can add indices to Chapel's sparse domains, just iterate over them and add to the domain using +=.

Challenge: Can you make it run fast at-scale?

Likely Answer: Hmmm....

A Quick Background on Copy Aggregation

- Copying random data into an ordered array is a common operation
    - sometimes called "gather"

```
forall (dst, idx) in zip(DstArr, SrcInds) do
  d = SrcArr[idx];
```

Results in random remote access

```
forall (d, idx) in zip(DstArr, SrcInds) with (var agg = new DstAggregator(int)) do
  agg.copy(d, SrcArr[idx]);
```

Random access is aggregated and data moved in bulk

Improving Arkouda's Sparse Linear Algebra Capabilities

Challenge: Can you create a distributed sparse domain & array pair,

- using 3 Arkouda pdarrays for rows, columns, and values,
- where the arrays are not necessarily sorted, nor contain unique data ?

Potential Answer: Well, of course! You can add indices to Chapel's sparse domains, just iterate over them and add to the domain using +=.

Challenge: Can you make it run fast at-scale?

Likely Answer: Hmmm....

Improving Arkouda's Sparse Linear Algebra Capabilities

Challenge: Can you create a distributed sparse domain & array pair,

- using 3 Arkouda pdarrays for rows, columns, and values,
- where the arrays are not necessarily sorted, nor contain unique data ?

Potential Answer: Well, of course! You can add indices to Chapel's sparse domains, just iterate over them and add to the domain using +=.

Challenge: Can you make it run fast at-scale?

Likely Answer: Hmmm.... A-ha! I am going to use copy aggregation!

Challenge: OK, can you copy the data in aggregate, and populate a sparse matrix during the operation?

Enter Custom Aggregation

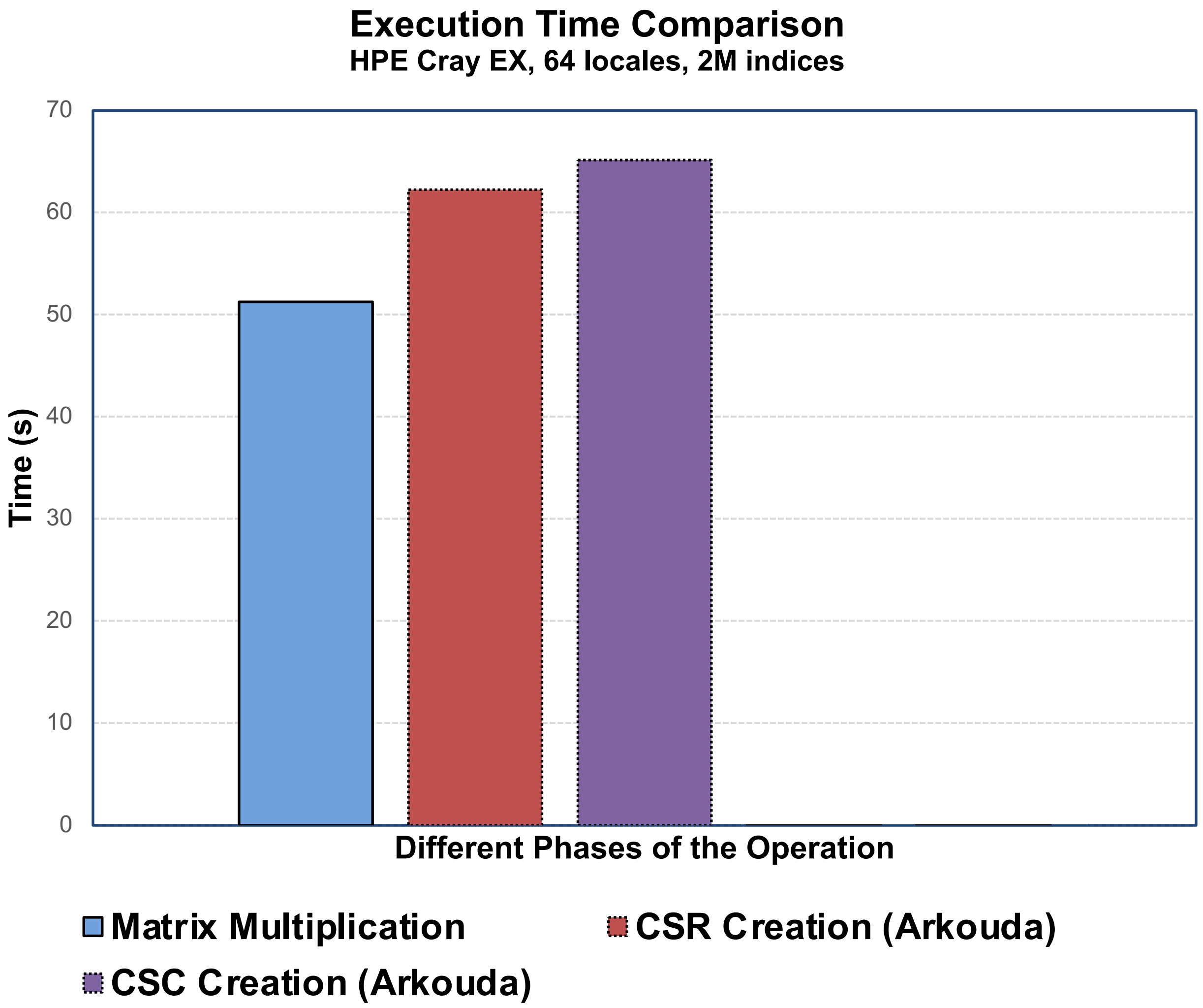
```
forall (i,j,v) in zip(rows,cols,vals) with (
    var agg = new CustomDstAggregator(
                  new shared SourceHandler(SparseDom,SparseArr)
              )
) {
  agg.copy((i,j,v));
}
```

Custom Aggregator allows any operation to be done after the data is moved

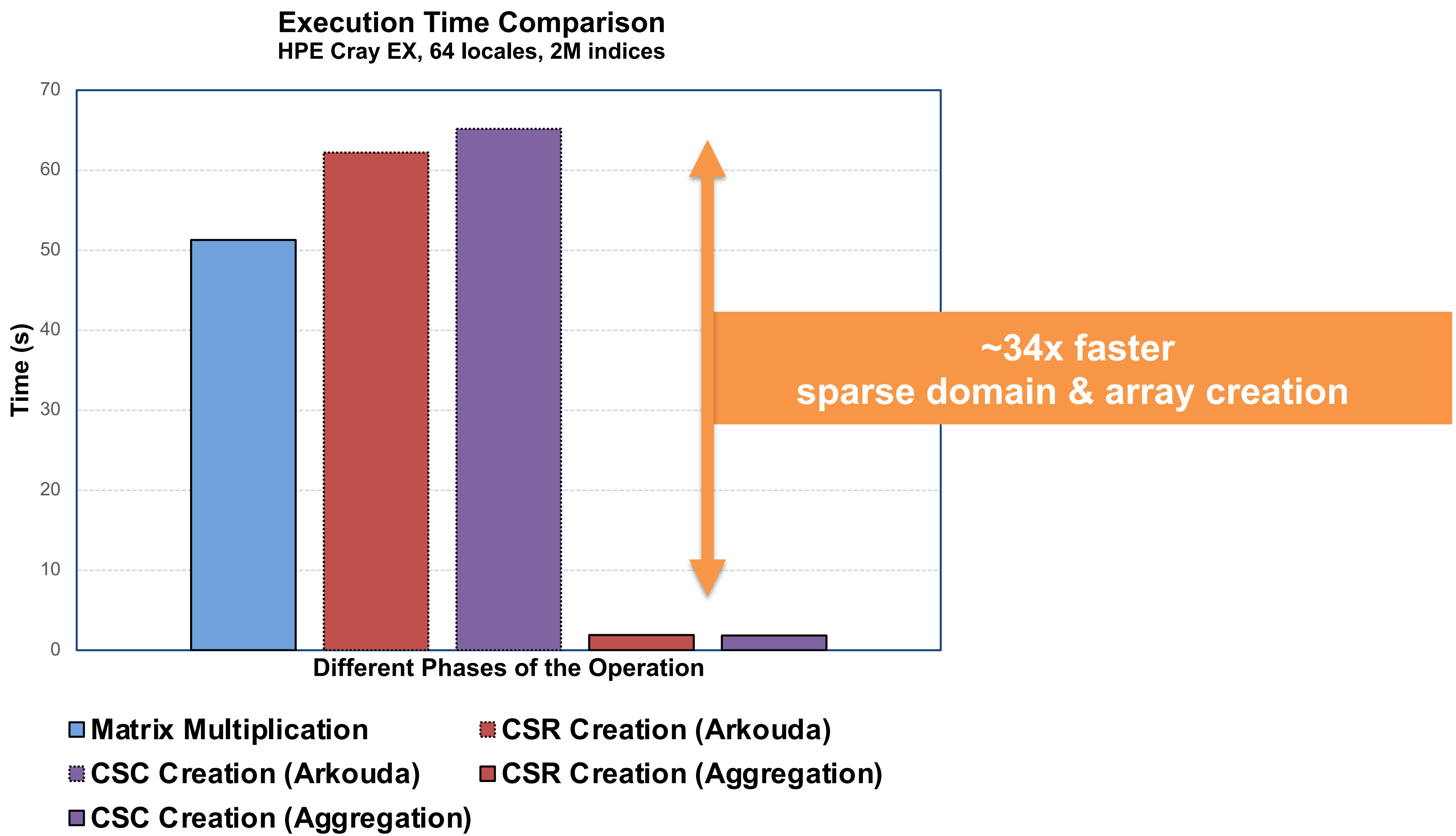
User-defined type to describe the operation

We are still working on finishing this effort

Performance w/o Custom Aggregation



Performance w/ Custom Aggregation



Honorable Mentions

Checkpointing

- Arkouda server's state can now be checkpointed (for the most part, we are still closing gaps)

```
ak.save_checkpoint("cp_name")   # arrays stored in server's symbol table are saved on the file system
ak.load_checkpoint("cp_name")   # and they are loaded back
```

- You can also opt-in for automatic checkpointing

```
> ./arkouda_server --checkpointMemPct=0.6 --checkpointIdleTime=300
```

Checkpoint after each operation if the used memory is >=60% of available memory

Checkpoint if the server is idle for 300 seconds

Python Interoperability

- Enables the user to run any simple Python function on Arkouda's pdarrays

```
arr = ak.array([1,2,3])
res = ak.apply(arr, lambda x: x+1)  # res is now [2, 3, 4]
```

Conclusion & Outlook

Outlook

αρκούδα
massive scale
data science

- **What's Next**
  - Complete per-function alignment with NumPy semantics
  - Deepen pandas-style functionality and DataFrame operations
  - Advance benchmarking, diagnostics, and tooling for developers
- **Get Involved**
  - Open-source and community-driven — new contributors welcome!
  - **13 active contributors** over the past year.
  - https://github.com/Bears-R-Us/arkouda
  - Help shape Arkouda's next phase through **code, docs, testing**, and **new use cases.**

Conclusion

αρκούδα
massive scale
data science

- **Arkouda in 2025**
  - Mature, **NumPy-like** framework for distributed analytics
  - Stronger **alignment with NumPy 2.0** and **pandas semantics**
  - Expanded **multi-dimensiona**l and **Python interop** support
  - Enhanced **random utilities**, **sparse matrices**, and **checkpointing**
  - Faster **parquet I/O**
  - Easier deployment via **Spack** and **Docker**, and **Kubernetes**
- **Key Takeaway**
  - Arkouda brings **Python's productivity** to **HPC scale** — enabling reproducible, data-intensive computing at interactive speed.

Thank You

Arkouda Contributors (Past Year)
@ajpotts • @drculhane • @1RyanK • @jade-abraham • @vasslitvinov
@ShreyasKhandekar • @e-kayrakli • @jeremiah-corrado • @jaketrookman
@stress-tess • @john-hartman • @lydia-duncan • @alvaradoo