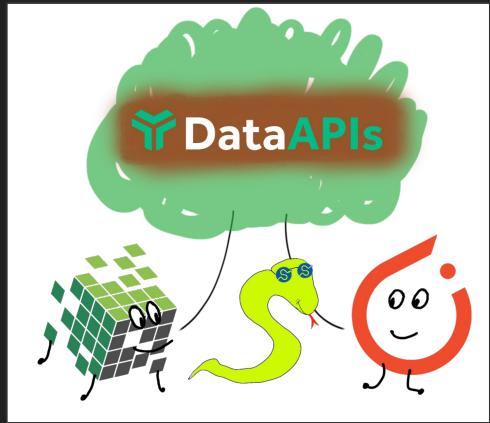
The Array API Standard in SciPy

Lucas Colley, EuroSciPy 2024

Thursday 29th August 2024, 11:30–11:50, Room 6,
Maritime University of Szczecin, Poland

These Slides

https://github.com/lucascolley/euroscipy24-slides



Why? Introduction to Interoperability

TODO:

- Alice data scientist - been gambling
- Holiday? No, shiny new GPU
- Why? Bob employee of the month, laughs at Alice and calls her slow
- We don’t like Bob
- Alice decides to try to make her workflow run on GPU

Why? Introduction to Interoperability

- GPU: NumPy -> CuPy
- Problem: SciPy CuPy support

```
>>> import scipy
>>> import numpy as np
>>> scipy.stats.{...}

>>> import cupy as cp
>>> scipy.stats.{...}
…
```

Why? Introduction to Interoperability

- Finds torch library with same functionality, so CuPy -> PyTorch
- Problem: API differences (missing torch functions)
- Gives up

Why? Introduction to Interoperability

- Alice’s questions: why can’t SciPy use CuPy arrays / work on GPU? Why don’t array libraries have a common API? Why does {library x} only work with one type of array?

About Me

- SciPy Maintainer (the algorithm library, not the conference!)
- Computer Science & Philosophy Undergraduate, Christ Church, University of Oxford
- Previously, an intern at Quansight Labs
- Based in Newcastle upon Tyne, UK

Why? Interoperability - SciPy

- SciPy is entirely built around NumPy arrays
- “Support for distributed arrays and GPU arrays” has been on SciPy’s roadmap for over 5 years.
- But “NumPy provides an array implementation that’s in-memory, CPU-only and single-threaded”[1].

[1] https://data-apis.org/array-api/latest/use_cases.html#use-case-scipy

Why? Interoperability - The Scientific Python Ecosystem

‘Array (provider) Libraries’

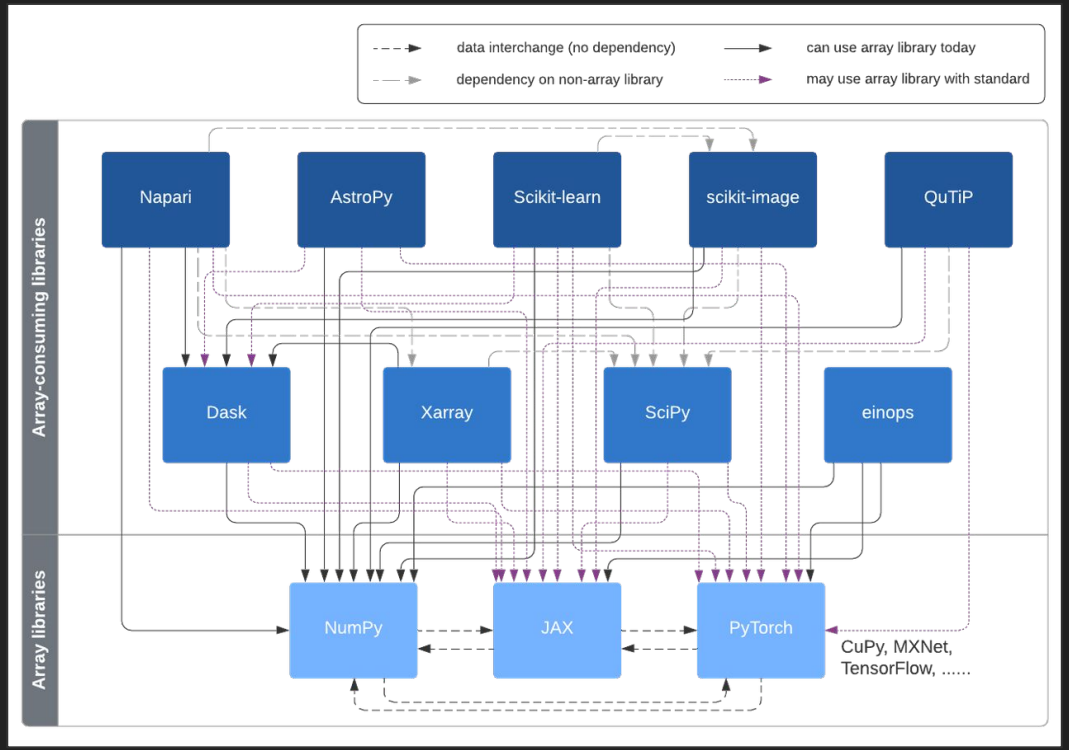
‘Array Consumer Libraries’

Credit: Aaron Meurer, ‘Python Array API Standard’, SciPy 2023

https://github.com/data-apis/scipy-2023-presentation/blob/main/presentation/Slides.pdf

Aside: It's Complex

A peek into the complex (actual and potential) interoperability within the Scientific Python Ecosystem (for illustrative purposes only):



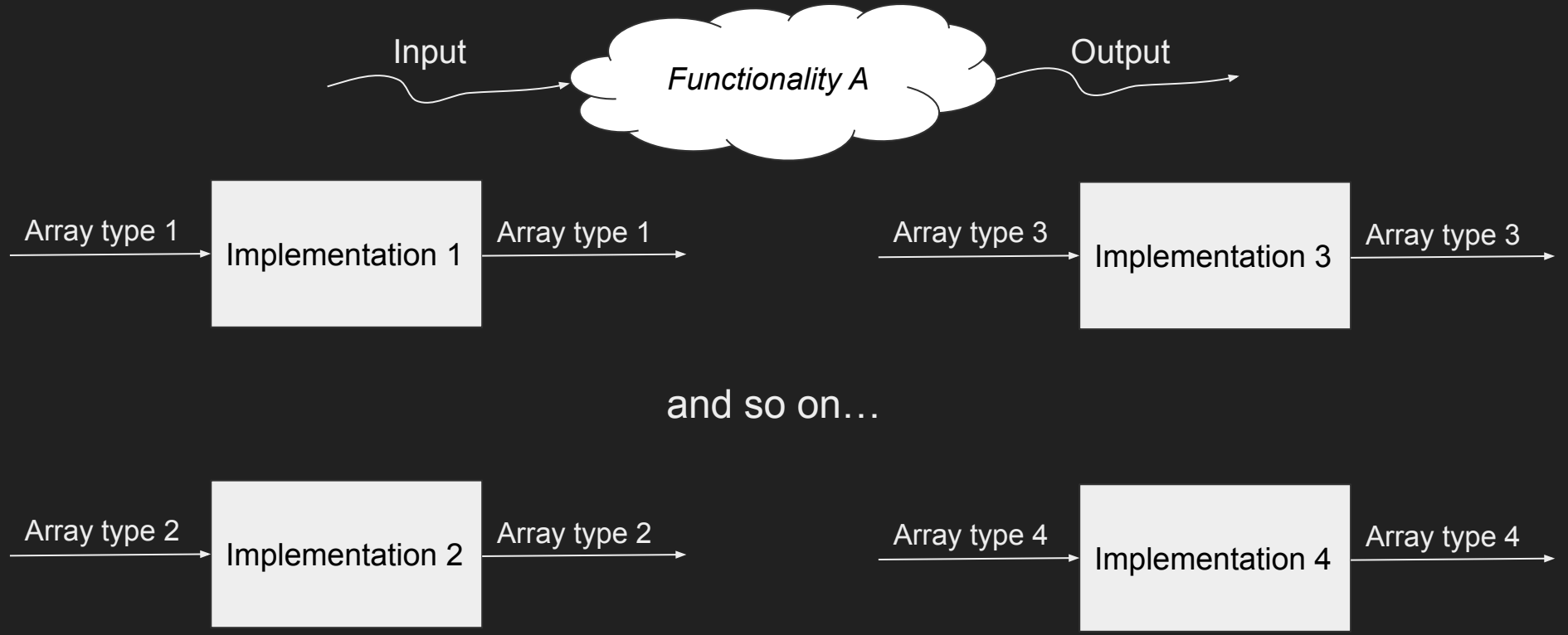
Credit: Ivan Yashchuk & Ralf Gommers, 'A vision for extensibility to GPU & distributed support for SciPy, scikit-learn, scikit-image and beyond', 2021
https://labs.quansight.org/blog/2021/11/pydata-extensibility-vision

Why? Interoperability - The Bigger Picture

- Layered but divided ecosystem
- Interoperability is about much more than just bringing GPU support to the NumPy ecosystem

Why? Interoperability - The Current Ecosystem

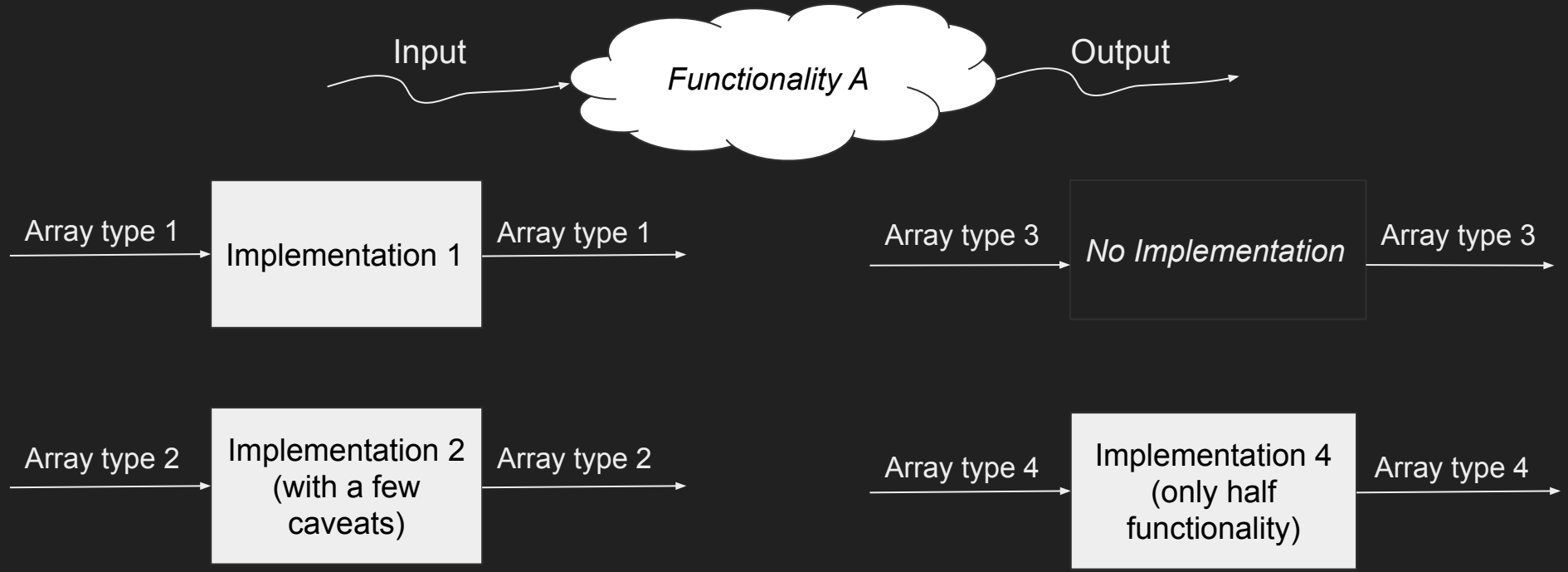
Why? Interoperability - The Current Ecosystem



Why? Interoperability - The Current Ecosystem

- Problem: maintenance cost!
- Difficult to make improvements in every implementation at once
- New array library? Needs another implementation
- All of this duplication does not seem efficient…

Why? Interoperability - The Current Ecosystem (really)



What? The (Python) Array API Standard

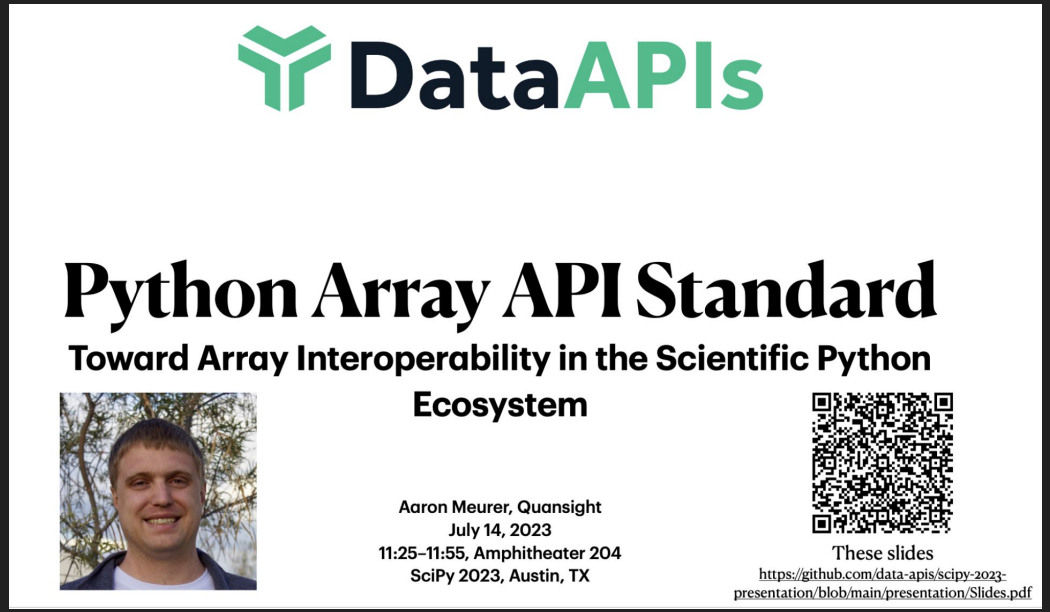
- Specifies a standard API for array libraries
- Starting from a minimal set of commonly implemented functionality
- Thus enabling “array-agnostic” implementations

- https://github.com/data-apis/array-api

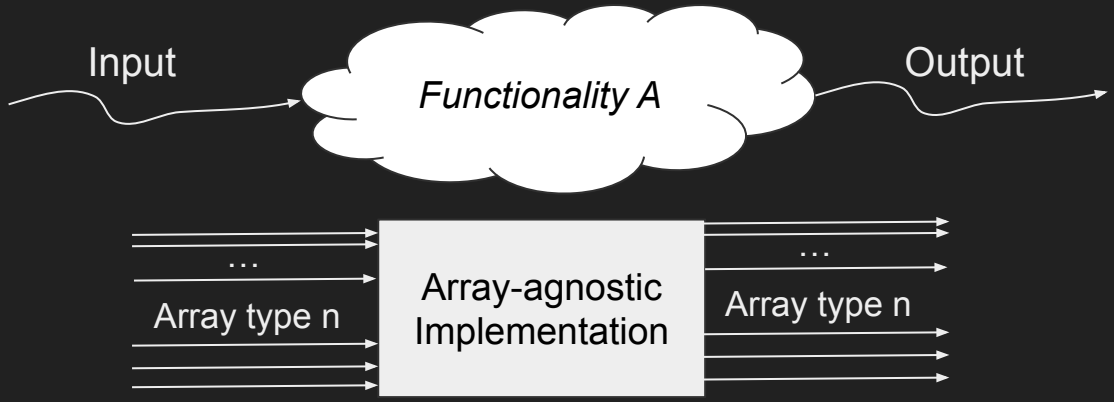
Aside: The Python Array API Standard

- A lot more to say: past work, design principles, testing, methodology

https://youtu.be/16rB-fosAWw

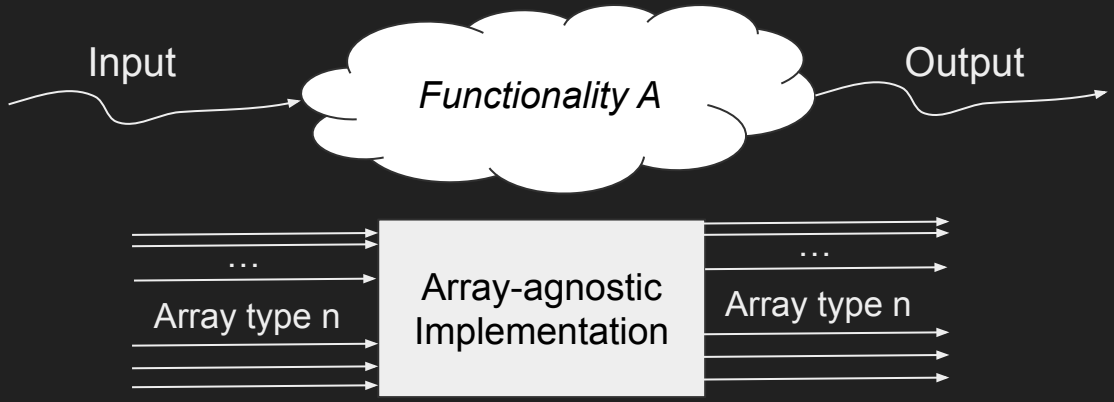


What? Interoperability - A Unified Ecosystem

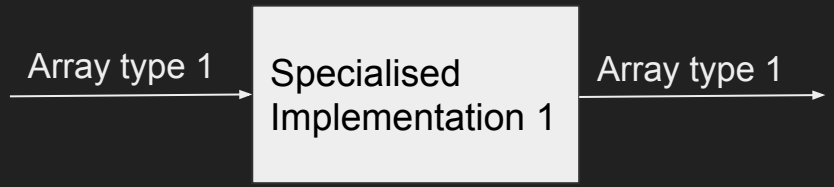


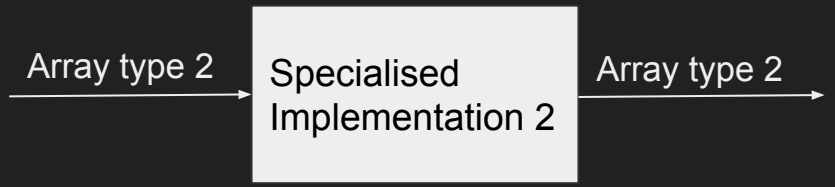
- Each array library implements the standard API
- Array-agnostic implementations which use just the standard API will work with all of them

What? Interoperability - A Unified Ecosystem



- Specialised implementations still possible (but only sometimes useful)





How? `xp = array_namespace(x)`

- The first line of most converted functions - get the namespace `xp` to use with the input arrays. Typically replace `np` with `xp`.

```
## scipy.cluster.vq.whiten
+ xp = array_namespace(obs)
+ obs = _asarray(obs, check_finite=check_finite, xp=xp)
- obs = _asarray_validated(obs, check_finite=check_finite)
+ std_dev = xp.std(obs, axis=0)
  zero_std_mask = std_dev == 0
+ if xp.any(zero_std_mask):
- if zero_std_mask.any():
      std_dev[zero_std_mask] = 1.0
      warnings.warn(...)
  return obs / std_dev
```

How? `x.__array_namespace__`

- `array_namespace` queries for the namespace via the dunder method
- No runtime imports! The array stores the namespace.
- `array_namespace` does other things for us:
  - Returns NumPy unless an env variable `SCIPY_ARRAY_API` is set
  - If it is set, reject some array types we do not want to support (e.g. `numpy.matrix`, arrays with esoteric dtypes, unknown objects which NumPy can't coerce with `np.asarray`)
- Simple to wrap and customise

How? array-api-compat

- Wraps each array library to (practically) full compliance
- Has an `array_namespace` helper which we wrap - so we can write code that works with array libraries today (despite missing the dunder method)
- Currently wraps NumPy, CuPy, PyTorch, JAX, Dask and Sparse

- https://github.com/data-apis/array-api-compat

How? Testing

- array-api-strict is a strict minimal implementation of the standard
- If our tests pass with array-api-strict arrays,
  they should pass with arrays from any compliant library
- We have pytest marks/fixtures to parametrize tests with array libraries & skip/xfail tests for certain backends/devices
- We have testing helpers which work across backends, such as `xp_assert_close` to replace `np.testing.assert_allclose`
- Tests that the input and output namespaces match are built-in to the helpers

- https://github.com/data-apis/array-api-strict

How? Testing

```
+ @skip_xp_backends(cpu_only=True)
+ @array_api_compatible
  @pytest.mark.parametrize("func", ['dct', 'dst', 'dctn', 'dstn'])
  @pytest.mark.parametrize("type", [1, 2, 3, 4])
  @pytest.mark.parametrize("norm", [None, 'backward', 'ortho', 'forward'])
- def test_fftpack_equivalience(func, type, norm):
+ def test_fftpack_equivalience(func, type, norm, xp):
      x = np.random.rand(8, 16)
+     fftpack_res = xp.asarray(getattr(fftpack, func)(x, type, norm=norm))
+     x = xp.asarray(x)
      fft_res = getattr(fft, func)(x, type, norm=norm)
-     fftpack_res = getattr(fftpack, func)(x, type, norm=norm)

-     assert_allclose(fft_res, fftpack_res)
+     xp_assert_close(fft_res, fftpack_res)
```

How? SciPy and Delegation

- Compiled code is out of scope for the Python array API standard
- But SciPy has a lot, due to being on the border of the array library and array-consumer library divide
- For some of `scipy.fft` and `scipy.linalg`, we can use the array API standard extensions
- There has been work on delegating to CuPy/PyTorch/JAX separately from the standard, for `scipy.special`, `scipy.ndimage`, `scipy.fft`, `scipy.signal`
- In practice, lots of overlap with array API standard work: `array_namespace`, testing
- If there is no library to delegate to, must convert to NumPy and back

Aside: Delegation, Dispatching

- But this is a different approach to interoperability - relies on other implementations existing
- Most important for code which needs or has big gains from specialised implementations in a compiled language
- Generalising to a dispatching mechanism is a challenge - c.f. Quansight-Labs/uarray, networkx, scientific-python/spatch

When and Where? SciPy - Current Progress

- Testing with PyTorch & JAX CPU in CI, CuPy also locally (GPU CI coming)
- Pure Python + NumPy code supports all libraries on all devices
- For compiled code, support for libraries with CPU execution
- Just a few rough edges - e.g. in-place operations, fancy indexing
- Modules currently covered: `cluster`, `constants`, `datasets`, `fft`, `io`, `ndimage`
- Partial coverage in `special`, `stats`
- Work in progress in various places

When and Where? SciPy - Looking Forward

- Still *a lot* of SciPy to convert!
- Support for `dask.array` in progress
- End goal: full coverage
- Then can think about raising warnings for array types which will see changed behaviour, and flipping the switch

When and Where? Open Source Contributors

- SciPy tracker
- Also scikit-learn
- A library you already contribute to!
- Feel free to direct maintainers to this talk

When and Where? Array Consumer Library Maintainers

- Your own library!
- Especially for lightweight, pure Python + array library code
- Can start experimenting now
- Reach out on the array-api-compat repo - can vendor or make a dependency
- Can look at the utilities SciPy and Scikit-learn are using
- Utilities which are generally useful will be upstreamed to array-api-compat in the future

When and Where? Array Library Maintainers

- Implement array API standard support in your main namespace!
  - `numpy`, `cupy`, `jax.numpy`, `sparse`, `ndonnx`
- Or contribute a wrapper to array-api-compat
  - `dask.array`, `pytorch` available now
- Feedback on specific parts of the array API standard:
  - https://github.com/data-apis/array-api
- Higher level feedback:
  - https://github.com/data-apis/consortium-feedback

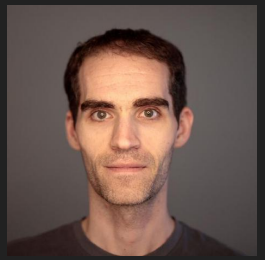
With thanks to:

Ralf Gommers
Chair, Consortium for Python Data API Standards; Chair, SciPy Steering Council

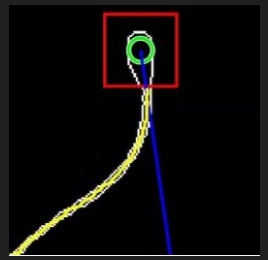
Aaron Meurer
Maintainer: array-api, array-api-compat, array-api-strict

Matt Haberland
SciPy Maintainer

Pamphile Roy
Original Author of SciPy's array API support machinery; SciPy Maintainer

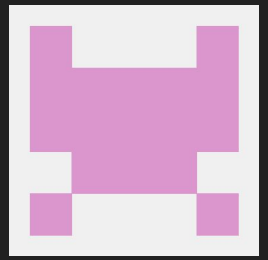


Tyler Reddy
SciPy Release Manager



Jake Bowhay
SciPy Maintainer

Evgeni Burovski
SciPy Maintainer



h-vetinari
SciPy Maintainer

Questions?

These Slides

https://github.com/lucascolley/euroscipy24-slides

SciPy Tracker

https://github.com/scipy/scipy/issues/18867

Array API Repo

https://github.com/data-apis/array-api